NPS ARCHIVE
1999.09
SKRETKOWICZ, S.

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CA 93943-5101

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

IMPLEMENTING A LOW-COMPLEXITY, ADAPTIVE, LAYERED VIDEO CODER FOR VIDEO TELECONFERENCING

by
Steven J. Skretkowicz

September 1999

Thesis Advisor: Murali Tummala
Second Reader: Robert E. Parker, Jr.

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1999	Master's Thesis

4. TITLE AND SUBTITLE
IMPLEMENTING A LOW-COMPLEXITY, ADAPTIVE, LAYERED VIDEO CODER FOR VIDEO TELECONFERENCING

5. FUNDING NUMBERS

6. AUTHOR(S)
Skretkowicz, Steven J.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 words)***

Real-time interactive video applications, such as video teleconferencing, present difficult challenges to network designers due to strict quality of service constraints and the limitations of traditional video compression schemes. These limitations reveal themselves notably in two areas: poor error robustness and a lack of flexibility when dealing with multicast scenarios over heterogeneous networks.

A more promising approach that improves error robustness while also offering a solution to the network heterogeneity problem is to employ a layered video codec. This thesis presents the implementation of a new layered video codec scheme. Block updating coupled with an aging algorithm is used in this scheme to select macroblocks for transmission. Block updating selects macroblocks that have changed due to scene motion, and the aging algorithm ensures that an entire frame is transmitted within a set time interval. Layering is accomplished through application of the fast Haar transform and/or the discrete cosine transform. Layer assignments are made by grouping bands of coefficients with similar variances. Quantization and encoding for motion video employs both an industry standard and uniform quantization with a custom variable length coding table. For static slides, uniform quantization and a second custom variable length coding table are employed. Rate control is accomplished via the reduction of a four-dimensional operational distortion surface to a one-dimensional optimal curve implemented as a simple table lookup of quantizers.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Video Teleconference, Layered Video Coder	125
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

ISN 7540-01-280-5500

Standard Form 298 (Rev.2-89)
Prescribed by ANSI Std. 239-18

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CA 93943-5101

Approved for public release; distribution is unlimited.

IMPLEMENTING A LOW-COMPLEXITY, ADAPTIVE, LAYERED VIDEO CODER FOR VIDEO TELECONFERENCING

Steven J. Skretkowicz
Lieutenant, United States Navy
B.S., Memphis State University, 1992

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 1999

SCHOOL

ABSTRACT

Real-time interactive video applications, such as video teleconferencing, present difficult challenges to network designers due to strict quality of service constraints and the limitations of traditional video compression schemes. These limitations reveal themselves notably in two areas: poor error robustness and a lack of flexibility when dealing with multicast scenarios over heterogeneous networks.

A more promising approach that improves error robustness while also offering a solution to the network heterogeneity problem is to employ a layered video codec. This thesis presents the implementation of a new layered video codec scheme. Block updating coupled with an aging algorithm is used in this scheme to select macroblocks for transmission. Block updating selects macroblocks that have changed due to scene motion, and the aging algorithm ensures that an entire frame is transmitted within a set time interval. Layering is accomplished through application of the fast Haar transform and/or the discrete cosine transform. Layer assignments are made by grouping bands of coefficients with similar variances. Quantization and encoding for motion video employs both an industry standard and uniform quantization with a custom variable length coding table. For static slides, uniform quantization and a second custom variable length coding table are employed. Rate control is accomplished via the reduction of a four-dimensional operational distortion surface to a one-dimensional optimal curve implemented as a simple table lookup of quantizers.

TABLE OF CONTENTS

I. INTRODUCTION 1

A. BACKGROUND 1

1. Multimedia Communication and Tactical Video Teleconference (VTC) ... 1
2. Video Compression and Robustness 2
3. Traditional Video Codecs 4
4. Receiver-Based Layered Multicast (RLM) 5

B. THESIS OBJECTIVE 6

C. THESIS ORGANIZATION 7

II. LAYERED VIDEO CODER DESIGN CONSIDERATIONS 9

A. VIDEO STREAM STRUCTURAL HIERARCHY 9

B. TRANSFORM CODING 10

1. Discrete Cosine Transform (DCT) 11
2. Discrete Wavelet Transform (DWT) 13
3. Comparison of Transforms 15

C. QUANTIZATION TECHNIQUES 16

1. Human Visual System (HVS) Weighting 17
2. Bit Allocation 17

D. ENTROPY ENCODING 18

E. FRAME CODING 19

F. MEASURING QUALITY 20

III. LAYERED VIDEO CODECS 23

A. LAYERED CODING SCHEMES 23

B. A LOW-COMPLEXITY, ADAPTIVE CODER FOR TACTICAL VTC 25

1. Temporal Compression via Block Updating 26
2. Aging Algorithm 29
3. Layering Strategy 32
4. Quantization and Entropy Encoding 41

5. Generating Customized VLC Tables 44
6. Rate-Distortion Relationship 45
7. Bit Rate Control 49
8. Scene Change Detection and Scene Type Determination 51

IV. RESULTS 53

V. CONCLUDING REMARKS 61

A. CONCLUSIONS 61

B. FUTURE WORK 62

APPENDIX A. MOTION VIDEO VARIABLE LENGTH CODING TABLE (VLC) 63

APPENDIX B. STATIC SLIDE VARIABLE LENGTH CODING TABLE (VLC) 69

APPENDIX C. MATLAB CODE LIBRARY 73

LIST OF REFERENCES 103

INITIAL DISTRIBUTION LIST 107

LIST OF FIGURES

Figure I-1: Simple VTC Multicast with Two Active and Two Passive Nodes. 2
Figure I-2: Overview of Layered Coder/Decoder. 5
Figure I-3: Adapting to Network Heterogeneity Using RLM. 6
Figure II-1: Basic Components of a Video Coder. 9
Figure II-2: Organizational Hierarchy for Compressed Video. 10
Figure II-3: Frequency Interpretation of DCT Coefficients. 13
Figure II-4: Structural Decomposition of Image Elements [4]. 13
Figure II-5: Octave-Based Wavelet Decomposition. 15
Figure II-6: HVS-Based Luminance Quantization Matrix [4]. 17
Figure III-1: Sample "Talking Head" Video Frame 24
Figure III-2: Basic Layered Video Coder Using the DWT. 24
Figure III-3: Functional Block Diagram of Proposed Layer Coder. 26
Figure III-4 Block Search Order; a.) Clockwise and b.) X-pattern. 28
Figure III-5: Second Order FHT Decomposition of a Macroblock 34
Figure III-6: Subband Variances after a First Order Analysis (Motion Video). 35
Figure III-7: Subband Variances after a Second Order Analysis (Motion Video) 36
Figure III-8: Partitions Resulting from Merge Algorithm (Motion Video). 37
Figure III-9 Final Layering Scheme for Motion Video Sequences. 38
Figure III-10: Subband Variances after a First Order Analysis (Static Slides). 39
Figure III-11: Subband Variances after a Second Order Analysis (Static Slides). 39
Figure III-12: Partitions Resulting from Merge Algorithm (Static Slides). 40
Figure III-13: Final Layering Scheme for Static Slide Sequences. 40
Figure III-14: Partitions after Merging Similar Non-Contiguous Partitions. 41
Figure III-15: Partitions for Quantizing Purposes (Static Slides). 41
Figure III-16: Quantization and Coding for Motion Video Macroblocks. 42
Figure III-17: Quantization and Coding for Static Macroblocks. 44
Figure III-18: Rate Distortion Curve with Possible Optimal Solution 46
Figure III-19: Operational Rate Distortion Curve (Motion Video) 47

Figure III-20: Reduced Operational Rate-Distortion Curve (Motion Video). 48
Figure III-21: Quantizer Table Triplet Values (Motion Video). 48
Figure III-22: Operational Rate-Distortion Curve (Static Slides). 49
Figure III-23: Operational Rate Control Curve (Motion Video) 50
Figure IV-1: Original and Reconstructed Frames from a Motion Video Sequence. 53
Figure IV-2: Original and reconstructed Frames from a Static Video Sequence 54
Figure IV-3: Bit Rates for (a) Fixed Quantization and (b) Bit Rate Control. 55
Figure IV-4: pSNR for (a) Fixed Quantization and (b) Bit Rate Control. 57
Figure IV-5: Comparison of Error Resilience. 58

LIST OF TABLES

Table I-1: Tactical VTC Multimedia Requirements. 7
Table III-1: Significance and Determination of DWT Subbands..... 33
Table III-2: Scan Order for Run-Length Encoding Quantized Coefficients. 43
Table IV-1: Rate Controlled and Uncontrolled Motion Video Sequence Statistics..... 57

LIST OF ABBREVIATIONS, ACRONYMS, AND SYMBOLS

a_1	first order FHT average analysis vector
B	bit allocation per frame
$\overline{B}$	average bit allocation per frame
β	bit rate control parameter
d_1	first order FHT detail analysis vector
D	distortion
ΔB_{inter}	bit allocation error (or deviation)
ΔQ_i	change in quantizer table entry
$\Delta\sigma^2$	variance comparison metric
f	frame rate
$f(i,j)$	pixel block
$F(u,v)$, F_{uv}	coefficient block
F_{quv}	quantized coefficient block
HH	subband via first order FHT analysis; retains diagonal detail of image
HHHH	subband via second order FHT analysis; retains diagonal detail of HH subband
HHHL	subband via second order FHT analysis; retains vertical edge detail of HH subband
HHLH	subband via second order FHT analysis; retains horizontal edge detail of HH subband
HHLL	subband via second order FHT analysis; retains lowpass content of HH subband
HL	subband via first order FHT analysis; retains vertical edge detail of image
HLHH	subband via second order FHT analysis; retains diagonal detail of HL subband
HLHL	subband via second order FHT analysis; retains vertical edge detail of HL subband
HLLH	subband via second order FHT analysis; retains horizontal edge detail of HL subband
HLLL	subband via second order FHT analysis; retains lowpass content of HL subband
K	maximum peak-to-peak pixel value
k_i	subband i
L	layer
LH	detail matrix via first order FHT analysis; retains horizontal edge detail
LHHH	subband via second order FHT analysis; retains diagonal detail of LH subband
LHHL	subband via second order FHT analysis; retains vertical edge detail of LH subband

LHLH	subband via second order FHT analysis; retains horizontal edge detail of LH subband
LHLL	subband via second order FHT analysis; retains lowpass content of LL subband
LL	average matrix via first order FHT analysis; retains lowpass content
LLHH	subband via second order FHT analysis; retains diagonal detail of LH subband
LLHL	subband via second order FHT analysis; retains vertical edge detail of LH subband
LLLH	subband via second order FHT analysis; retains horizontal edge detail of LH subband
LLLL	subband via second order FHT analysis; retains lowpass content of LL subband
M	total number of macroblocks in a frame
$\overline{M}$	average number of macroblocks chosen per frame in test video sequences
$\overline{M}_{age}$	average number of macroblocks chosen per frame by the aging algorithm
N_b	Total number of subbands created by recursive application of FHT
P	Partition
q_1	first quantizing parameter
q_2	second quantizing parameter
q_3	third quantizing parameter
Q_{uv}	quantizer step size at position (u,v)
R	bit rate
R_c	bit rate constraint
R_o	arbitrary bit rate
R_{target}	target bit rate
σ^2	variance
σ^2_{max}	maximum variance of all subbands
σ^2_{min}	minimum variance of all subbands
x	original image
$\hat{x}$	reconstructed image
x_0	original data vector with regard to FHT analysis
$x_{n,m}$	pixel value in current block
$x^R_{n,m}$	pixel value in reference block

1-D	One Dimensional
2-D	Two Dimensional
3-D	Three Dimensional
4-D	Four Dimensional
ASD	Absolute Sum of Differences

ATM	Asynchronous Transfer Mode
bpf	bits per frame
bpp	bits per pixel
CELP	Code Excited Linear Prediction
DCT	Discrete Cosine Transform
DPCM	Differential Pulse Code Modulation
DWT	Discrete Wavelet Transform
EOB	End-Of-Block
FHT	Fast Haar Transform
fps	frames per second
GOB	Group of Macroblocks
HVS	Human Visual System
IP	Internet Protocol
IT-21	Information Technology for the 21st Century
ITU	International Telecommunications Union
ITU-T	ITU Telecommunications Standardization Sector
JPEG	Joint Photographic Experts Group
kbps	kilobits per second
KLT	Karhunen-Loeve Transform
LAN	Local Area Network
M-JPEG	Motion-JPEG
MPEG	Moving Pictures Experts Group
MSE	Mean Squared Error
pSNR	peak Signal-to-Noise Ratio
QCIF	Quarter Common Intermediate Format
QoS	Quality of Service
RLE	Run-Length Encoding
RLM	Receiver-Based Layered Multicast
rt-VBR	real-time Variable Bit Rate
SAD	Sum of Absolute Differences
SNR	Signal-to-Noise Ratio
UD	Uniquely Decipherable
VLC	Variable-Length Code or Variable Length Coding
VOD	Video On Demand
VTC	Video Teleconferencing

ACKNOWLEDGEMENT

The work presented in this thesis is the culmination of support from many people. I am especially fortunate to have had the opportunity to work with a most dedicated advisor, Murali Tummala. He was instrumental in the completion of this work, and I am grateful for his guidance and insight. I have also had the honor and pleasure to work with another outstanding individual, Robert (Bob) E. Parker, Jr. Throughout the entire process of this work from concept to completion, Bob was forever unselfish with his valuable time. Finally, I acknowledge the support of my wife, Karen, and children, Elizabeth and Christopher. Words cannot express the gratitude I have for their patience and love.

I. INTRODUCTION

A. BACKGROUND

Video teleconferencing (VTC) is expected to contribute significantly to the US Navy's Information Technology for the 21st Century (IT-21) initiative. IT-21 seeks to change the paradigm to warfighting from a platform-centric approach to a network-centric approach, where information superiority is leveraged with smart weapons in order to achieve the desired result more effectively. The goal of IT-21 is to link all US Forces together in a network that enables transmission of voice, video, and data from individual workstations seamlessly to both local and remote users [1][2]. Employing VTC over a tactical network at the battle group level caters to several useful applications, such as collaborative planning, distance learning, remote maintenance and telemedicine.

1. Multimedia Communication and Tactical Video Teleconference (VTC)

In general, multimedia communications are either unicast or multicast. Unicast represents peer-to-peer communications while multicast represents *m* to *n* communications where *m* ranges from 1 to *n*. Unicast includes any client-server applications, such as video on demand (VOD) or IP telephony. Multicast examples include distance learning and remote conferencing. The tactical VTC scenario considered here is inherently a multicast application running over a heterogeneous, wireless network. Here, "heterogeneous" implies that a connection traverses a series of links (each imposing potentially different bandwidths), and the recipient workstations differ in capability and capacity; "wireless" implies an internetwork of wireline local area networks (LANs) connected by at least one wireless channel. As such, each transmitter transmits to multiple receivers in the multicast group. The multicast group consists of some combination of active participants that are allowed to transmit and passive participants that only receive. For example, the problem of transmitting multicast,

multimedia traffic over ATM networks was examined in [3]. This situation is illustrated in Figure I-1.

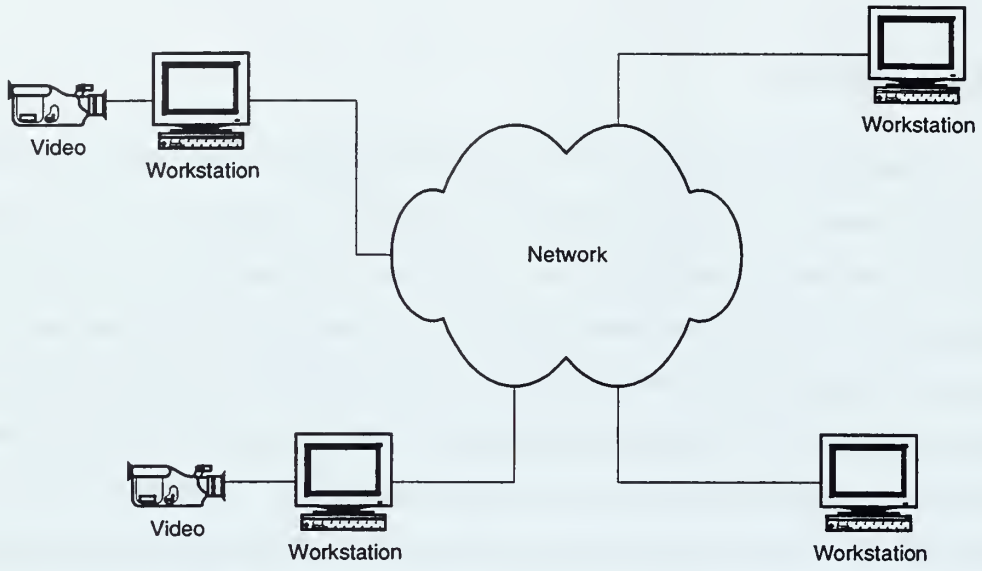

Figure I-1: Simple VTC Multicast with Two Active and Two Passive Nodes.

While the network described here affords large bandwidth within each shipboard LAN, VTC realizes its true value only if implemented within the entire battlegroup. It is because the wireless channel constrains both the bit rate and the bit error rate (each of which, in turn, impacts the robustness of the VTC application) that the tactical VTC network presents challenges not encountered with wireline networks and provides a hostile environment for traditional video codecs.

2. Video Compression and Robustness

Since video and audio, to a lessor degree, are bandwidth intensive, the signals must undergo video coding prior to transmission, trading a reduction in bandwidth for a reduction in quality. Because the human visual system (HVS) places greater relative importance on lower frequencies than higher frequencies, typical motion video is perceived as primarily lowpass [4]. Therefore, lossless two-dimensional (2-D) transform methods are used to create a frequency domain representation of an image. The HVS perceptive properties can then be exploited by quantizing the resulting coefficients to varying degrees of precision with higher precision allotted to the lower frequency

coefficients. Quantization reduces the dynamic range of the coefficients and loses information, but it enables fewer bits to represent the coefficients. Usually, many of the higher frequency coefficients are zeroed out via this quantizing process; runs of zeroes are created. Because zeros need not be explicitly represented, run-length encoding (RLE) is utilized to generate a more compact representation of the quantized coefficients, which is subsequently replaced by a more efficient lossless variable length code (VLC). These techniques are collectively called spatial compression. Spatial compression is the basis of image compression standards like that of the Joint Photographic Experts Group (JPEG).

A video codec may simply treat each frame of a video as a separate still image and subject it to spatial compression independently from the other frames. This approach is known as intraframe coding. An example is Motion-JPEG (M-JPEG). Intraframe coding has the advantage of superior error resilience since decode errors are confined strictly to the current frame. Its disadvantage is that compression gain with acceptable image quality is limited to approximately 0.5 bits per pixel (bpp) [4].

For a given quality, higher compression gains are realized if the video codec exploits the high degree of correlation that video frames tend to exhibit on a frame-to-frame basis. This is called interframe coding, and it eliminates redundancy by coding only the differences in successive frames. The compression gains achieved by interframe coding vary in relation to the degree and type of motion that occurred between successive frames. Nearly static-content frames exhibit high correlation and result in high compression gain while highly dynamic-content frames have little correlation and result in less compression gain. If two successive frames have no correlation, perhaps due to a scene change, interframe coding performs no better than intraframe coding and typically performs worse due to the overhead required to track motion. The disadvantage to interframe coding results from the dependency between frames at the decoder. If errors occur in the current frame, the errors tend to propagate among frames as well as spatially within frames [5][6]. Consequently, video codecs like that of the Moving Pictures

Experts Group (MPEG) incorporate both types of compression techniques in an attempt to increase compression gain and bound error propagation.

3. Traditional Video Codecs

Low bit rate video coding standards such as H.261 and H.263 perform best in homogeneous, unicast environments. The video server negotiates a desired Quality of Service (QoS) consistent with the desired video quality and available bandwidth prior to delivery. Since network conditions are rarely static, the received video quality typically changes due to dropped or incomplete frames caused by losses within the network. With the implementation of a closed loop control scheme via feedback reports from the recipient, the server can react to the changing network conditions and adjust the quantization, frame size, or frame rate in order to vary the bit rate.

However, when traditional video codecs are applied in a wireless, multicast, heterogeneous environment, shortcomings are revealed. First, the traditional scheme relies on guaranteed bandwidth for delivery, trading bandwidth for quality. Selecting an appropriate quality (and therefore required bandwidth) poses a dilemma in a heterogeneous network. Since each user is reached by a different path on the network, each experiences different levels of congestion. Even with feedback, the controllable application is faced with a quandary in determining how to make adjustments. Sending high quality, high bandwidth video supports some users but leaves low bandwidth recipients with degraded video due to high packet loss. If the lowest common denominator is supported instead, all recipients are forced to view lower quality video, and the high bandwidth links are underutilized. Clearly, meeting the varied expectations with a single video stream is impractical and transmitting multiple video streams with gradations in quality demands a much greater bandwidth expense.

Second, the poor error robustness demonstrated by traditional low bit rate video coders is especially troublesome since retransmission is not practical in a real-time application. Finally, feedback itself is undesirable in a low bit rate network. Feedback employed with the goal of mitigating congestion actually consumes available bandwidth, causes an additional load on constrained nodes, and increases congestion further.

4. Receiver-Based Layered Multicast (RLM)

A promising approach that offers a solution to the network heterogeneity problem while offering some improvement in error robustness is the receiver-based layered multicast (RLM) scheme proposed by McCanne et al. [7]. RLM employs a layered video codec that transmits video in scalable layers that progressively refine quality. An independently decodable base layer is generated that guarantees a minimum acceptable quality. Separate enhancement layers increase quality in a hierarchical manner. With RLM, each recipient can decode just the base layer for low, but acceptable, quality or add one or more enhancement layers to improve quality as bandwidth and hardware permit. This idea is illustrated in Figure I-2.

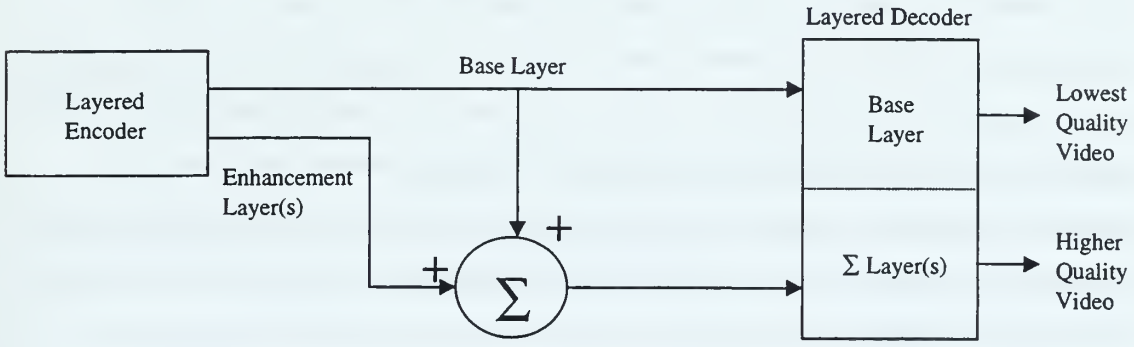

Figure I-2: Overview of Layered Coder/Decoder.

RLM represents a starting point for designing an integrated approach to improving robustness. The layered structure slightly reduces the effects of congestion because a particular node only needs to carry subscribed layers. Unsubcribed layers can be dropped. Figure I-3 illustrates this concept. Furthermore, earlier work by Rhee and Gibson indicates that layered video exhibits improved resilience to bit errors introduced during transmission because spreading bit errors across multiple layers has less negative impact on the reconstructed video [8].

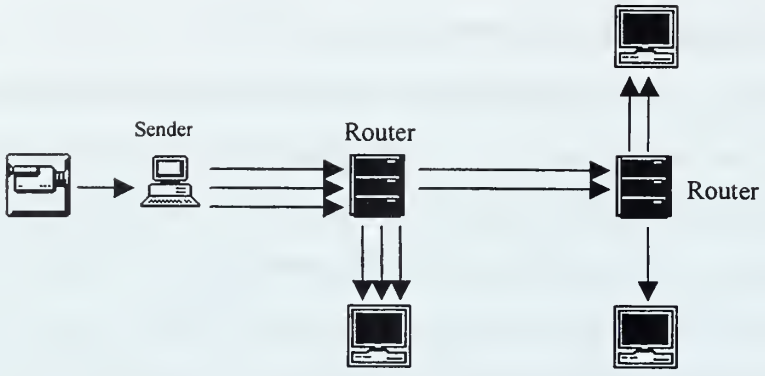

Figure I-3: Adapting to Network Heterogeneity Using RLM.

B. THESIS OBJECTIVE

The main objective of this thesis is to implement in Matlab[1] the new layered video coding scheme for a tactical VTC proposed by Parker [9] [10] for use in multicast, hetergeneous, wireless, asynchronous transfer mode (ATM) networks. The tactical VTC session is assumed to consist of low-motion video, such as a "talking head" in the style of a typical newscast and static displays in the style of an overhead presentation. No assumption is made about the relative proportions of the two types of content; the session could be all low-motion video, a sequence of static displays, or any combination of the two types in any order.

The complete specifications of the coder are detailed in Table I-1 [9]. For the implementation in this thesis, a color depth of 8-bit grayscale is evaluated, but the technique can be expanded to include 4:2:0 sub-sampled 24-bit color. Audio compression to 8 kbps or less may be accomplished via code excited linear prediction (CELP) speech coding, but it is not addressed further in this work. Error robustness is provided via a block updating scheme that limits the impact of decoder errors. Layering is accomplished by judiciously grouping the frequency domain content obtained from the fast Haar transform (FHT) and/or the discrete cosine transform (DCT); the exact method of frequency decomposition depends on the video content. Handling both motion video

[1] Matlab is a registered trademark of The MathWorks, Inc.

and static slides with a single coder requires significant flexibility and compromise since the frequency characteristics of each are different. Therefore, the coder is optimized to handle each type of content separately – separate layering and separate, custom VLC tables. Finally, the bit rate control issue is examined, and an approach that reduces an *n*-dimensional rate control problem to a simple table lookup is implemented.

VTC Stream	Parameter	Value
Video	Bandwidth	64-96 kbps
	Resolution	176×144 (QCIF)
	Frame Rate	10 fps
	Color Depth	8-bit gray/4:2:0 24-bit color
Audio	Bandwidth	< 8 kbps

Table I-1: Tactical VTC Multimedia Requirements.

C. THESIS ORGANIZATION

Chapter II considers techniques for coding video. A general discussion of video stream structural hierarchy, transform coding methods, quantization techniques, entropy encoding, frame coding, and quality measurement is presented. Chapter III presents the specific techniques utilized in coding both motion video and static slides. Chapter IV presents results. Conclusions and recommendations for future study are given in Chapter V. Appendix A and Appendix B provide the custom VLC tables used with motion video and static slides, respectively. Appendix C is a Matlab code library of the layered video coder implementation.

II. LAYERED VIDEO CODER DESIGN CONSIDERATIONS

The basic components of a video coder are shown in Figure II-1. This chapter begins with an explanation of video stream hierarchy and then discusses each of the component parts displayed in Figure II-1. Motion compensation is present only if the coder attempts to exploit frame-to-frame correlation. Otherwise each frame is processed independently. The chapter concludes by addressing a method for quantifying the amount of distortion introduced by the processing.

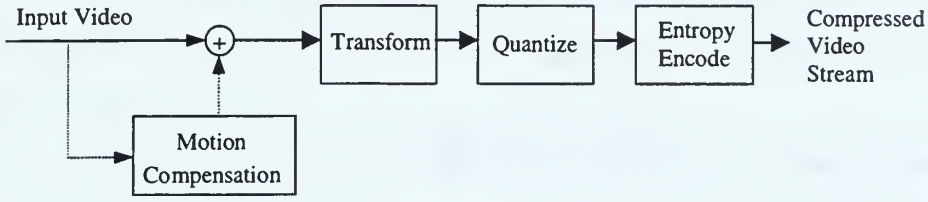

Figure II-1: Basic Components of a Video Coder.

A. VIDEO STREAM STRUCTURAL HIERARCHY

A video stream is organized into a hierarchy of logical components. Although the specific organizational scheme varies with the particular coder under consideration, some of the most common components are the following.

A frame or picture is the basic display unit. It is a sampled version of the original scene taken at a particular instant in time. Each frame is composed of a rectangular array of pixels or points within the frame. Each pixel contains a data structure indicating its luminosity or color. The dimensions of the array define the picture resolution, given as columns × rows, such as the ITU-T defined QCIF resolution of 176×144. Pixels within a frame are organized, in order of increasing size, into blocks, macroblocks, and groups of macroblocks (GOBs). A block is an 8×8 array of pixels; a macroblock is a 16×16 array of pixels or, equivalently, four blocks. A frame may be considered as rows of

macroblocks. One or more contiguous rows of macroblocks are termed a GOB. This hierarchy is illustrated in Figure II-2.

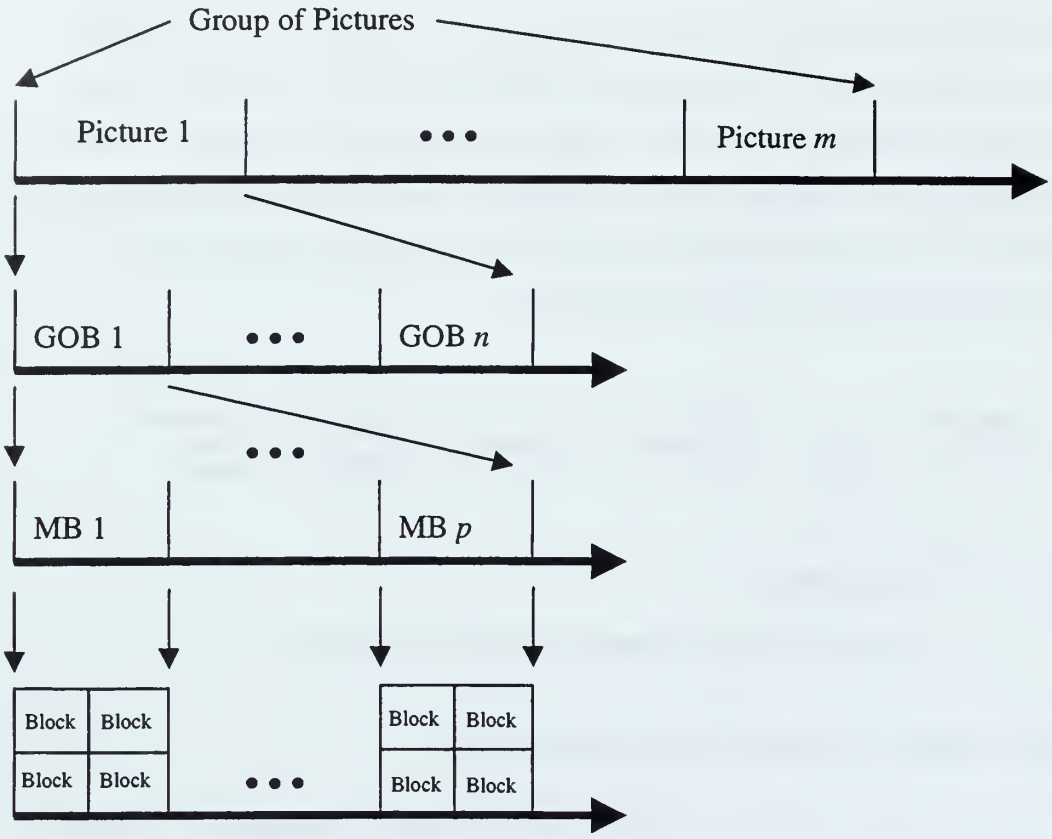

Figure II-2: Organizational Hierarchy for Compressed Video.

B. TRANSFORM CODING

Although compression through direct scalar quantization of pixel values is possible, it is inefficient. An alternative approach is to use transform coding. Because contiguous pixels within a frame tend to be highly correlated, the application of a suitable linear transform[2] to decorrelate the pixels yields two primary advantages. The first is a property termed "energy packing" efficiency. The second is that the resulting coefficients are more conducive to perceptual-based quantization schemes.

[2] The transform should be lossless and invertable. Lossless means that no information is lost through application of the transform. Invertable means that the original information is recoverable.

A signal is decorrelated if the application of a transform causes the signal's autocorrelation matrix to become diagonal; that is, it uncorrelates the resulting coefficients. The optimal transform tightly packs all the energy (information) into the minimum number of coefficients possible, resulting in the highest energy packing efficiency. Arranging these N coefficients in decreasing order of magnitude and retaining only the first k coefficients gives the least distortion as measured by mean squared error (MSE) compared to any other set of k coefficients. Similarly, a given level of quantization of the decorrelated coefficients results in the least distortion of the original data [11]. Because certain transform coefficients may hold greater perceptual relevance by the HVS, this dependency can be exploited by utilizing a frequency-based transform and then quantizing– a lossy process – with a step size proportional to the perceptual importance of each coefficient.

1. Discrete Cosine Transform (DCT)

Theoretically, the discrete-time Karhunen-Loeve transform (KLT) provides the greatest energy packing efficiency [11]. However, two liabilities preclude the use of the KLT in video compression. First, the KLT is extremely computationally intensive – requiring order N^2 operations. Second, the KLT is signal dependent – requiring a separate eigenvector calculation for each transformed data block. Instead, transforms that approach the energy packing efficiency of the KLT and possess more efficient algorithms are utilized in video coders.

A widely used transform for image processing is the two-dimensional (2-D) discrete cosine transform (DCT). The 2-D DCT provides energy packing performance very close to that of the KLT and can be implemented with fast algorithms that reduce the computational effort to the order of $N\log_2 N$ [4]. A frame is transformed by partitioning the frame into $N\times N$ regions of pixels and applying the 2-D DCT to each individual region. N can be any integer provided that integer multiples of N equals the overall dimensions of the frame. Because the correlation among contiguous pixels tends to decrease with increasing size of the $N\times N$ region which, in turn, decreases the resulting compression gain, the typical $N\times N$ size used with the 2-D DCT is 8×8 (a block).

Denoting the original block as *f(i,j)* and the coefficient block by *F(u,v)*, the 2-D DCT is given by [4]

$$F(u,v) = \frac{2}{N} C(u)C(v) \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} f(i,j) \cos\left(\frac{(2i+1)\pi u}{2N}\right) \cos\left(\frac{(2j+1)\pi v}{2N}\right), \qquad \text{(II-1)}$$

where

$$C(x) = \begin{cases} \frac{1}{\sqrt{2}}, & x = 0 \\ 1, & \text{otherwise.} \end{cases} \qquad \text{(II-2)}$$

The inverse DCT is given by

$$f(i,j) = \frac{2}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} C(u)C(v) \cos\left(\frac{(2i+1)\pi u}{2N}\right) \cos\left(\frac{(2i+1)\pi v}{2N}\right) F(u,v). \qquad \text{(II-3)}$$

The result is a block of 64 coefficients having a spatial frequency interpretation as shown in Figure II-3. The *F(0,0)* coefficient represents the DC contribution and the remaining 63 coefficients represent the AC contribution. The different elements of an image map into the frequency domain as shown in Figure II-4 [4]. The application of the 2-D DCT to an 8×8 block, where there is typically little variation from pixel to pixel, leads to a predominance of lowpass content in the frequency domain. Given this condition, the magnitudes of the AC coefficients are largest in the region around the DC coefficient and diminish with increasing spatial frequency.

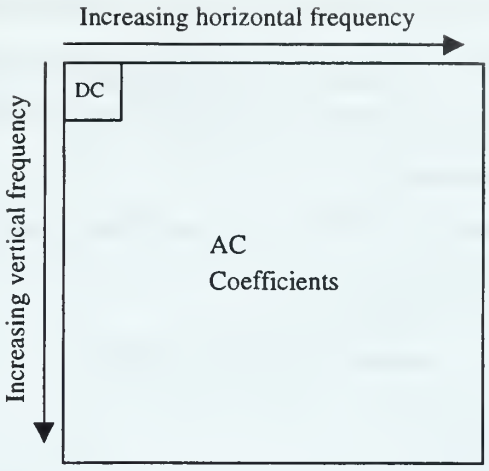

Figure II-3: Frequency Interpretation of DCT Coefficients.

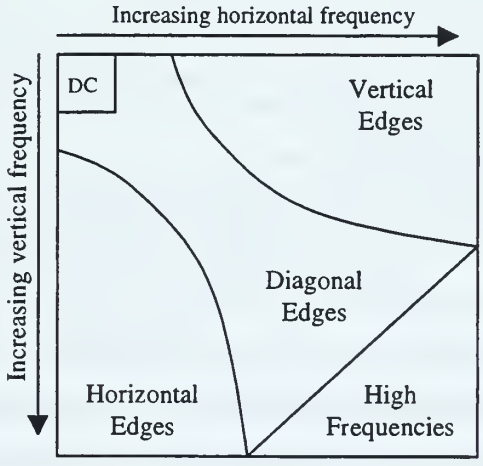

Figure II-4: Structural Decomposition of Image Elements [4].

2. Discrete Wavelet Transform (DWT)

Wavelet coding is another technique for compressing still images and has been shown to offer slightly better image quality than DCT-based schemes for similar levels of compression at the cost of greater computational complexity [12]. Applying a wavelet transform to an image transforms that image into a "multifrequency component representation," where each component has its own frequency characteristics and spatial features.[3] A discrete wavelet transform (DWT) filters and decimates an image into

[3] The wavelet transform, like the 2-D DCT, is lossless and invertable.

separate subbands of coefficients containing a mixture of the high frequency and low frequency details. Image decomposition is accomplished via two analysis filters. The first extracts the low frequency content, the signal average; the second extracts the high frequency content, the signal details.

The simplest example of a DWT is the fast Haar transform (FHT) [13]. It is described by its signal average equation,

$$a_1(n) = \frac{1}{2}(x_0(2n) + x_0(2n+1)) \qquad \text{(II-4)}$$

and its signal detail equation,

$$d_1(n) = \frac{1}{2}(x_0(2n) - x_0(2n+1)), \qquad \text{(II-5)}$$

where x_0 is the original data vector, and vectors a_1 and d_1 are the first order analysis (decomposition) vectors. If x_0 contains 2^L elements, applying Equations II-4 and II.5 generates analysis vectors of length 2^{L-1}.

Referring to Figure II-5, the LL subband is calculated by applying the average equation to the columns and rows of the image. The LL subband retains the lowpass information from the original image and presents a coarse representation of the original image. Because typical images have lowpass characteristics, most of the energy from the image is represented in this subband. The HL subband is calculated by applying the average equation to the columns of the image and the detail equation to the rows. The HL subband retains the vertical edge details from the original image. Typically, less energy from the original image is represented in the HL subband compared to that of the LL subband. The LH and HH subbands are found analogously with LH retaining horizontal edge details and HH retaining the diagonal edge details. The HH subband typically has the lowest energy content. In fact, in some applications the HH subband is discarded in its entirety [14].

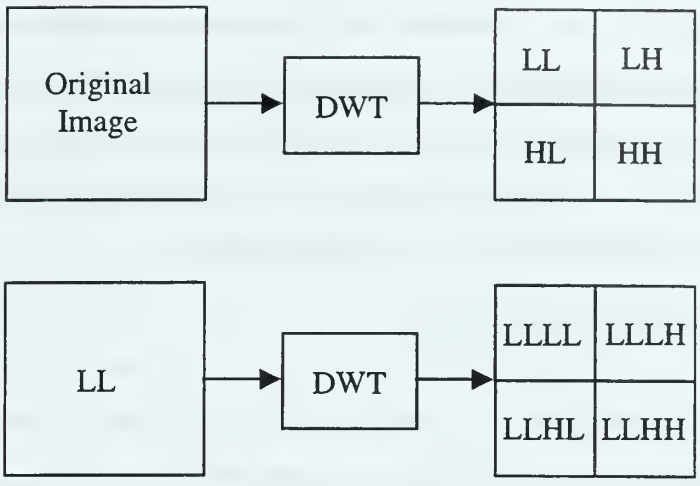

Figure II-5: Octave-Based Wavelet Decomposition.

In order to differentiate the frequency content of an image further, a second DWT can be applied. Still referring to Figure II-5, the effect of a second order octave-based decomposition obtained by applying the same analysis filters to the LL subband is illustrated. This increase in the number of subbands allows tailoring additional stages in the coder to emphasize the perceptually important details over less perceptual background noise as will be discussed in greater detail later.

3. Comparison of Transforms

Since the 2-D DCT is applied to pixel blocks within a frame that are subsequently quantized, it has a tendency to create "blocking" artifacts that disturb the continuity of the reconstructed frame. The same effect also leads to the presence of "ringing" artifacts around sharp edges. In contrast, the wavelet transforms typically are applied to the entire image and separate the entire image into regions of high and low frequency content. These regions may be quantized and coded independently and result in more efficient bit allocation. This produces a more visually pleasing, smoother reconstructed image compared to that obtained via a 2-D DCT at a comparable peak signal-to-noise ratio (pSNR). In general, at comparable pSNRs, wavelet transform coders offer compression gains superior to that of DCT-based coders [12].

However, several drawbacks have limited the utility of wavelet-based video compression. Wavelets achieve quality superior to DCT-methods by processing the

entire image or frame, but traditional video coders exploit temporal correlation at a subframe (usually macroblock) level. Although the error block could be transformed via a DWT, no significant advantage has been determined over the DCT, and the computational effort is greater [12]. However, the frequency decomposition offered by DWTs provides a powerful basis for layered video coding.

C. QUANTIZATION TECHNIQUES

After the transform, the coefficients are quantized. Quantization is a lossy process used to reduce precision and zero out some coefficient values. The benefit is that each coefficient can then be represented with fewer bits. However, the information loss is not recoverable, and the loss is manifested as distortion within the reconstructed image. This distortion is termed quantization noise.

The typical quantization scheme, uniform quantization, involves dividing each coefficient F_{uv} by the quantizer step size Q_{uv} and rounding the result to the nearest integer to produce a quantized coefficient F_{quv} as follows [15]:

$$F_{quv} = \text{nearest integer}\left(\frac{F_{uv}}{Q_{uv}}\right). \qquad \text{(II-6)}$$

The values used in image reconstruction are then F_{quv} multiplied by the step size. However, as Equation II-6 implies, the step value may be adjusted for each particular coefficient with Q_{uv} now representing elements of a quantizer matrix. Choosing the appropriate step size involves a trade-off between acceptable error and desired compression. Employing a small step size yields low quantization noise but little compression. The opposite is the case for a large step size. Although uniform quantization is often used, the choice is suboptimal since the individual coefficients are not distributed uniformly [16]. However, two broad approaches are employed to refine the selection of step size: HVS weighting and bit allocation strategies. HVS weighting heuristically refines the step size based on perceptual relevance while bit allocation strategies attempt to spread errors optimally across all coefficients.

1. Human Visual System (HVS) Weighting

Because the HVS places greater relative importance on lower frequencies than higher frequencies, step sizes based on HVS modeling are selected such that lower frequencies coefficients are quantized more finely while higher frequency coefficients are quantized more coarsely [4]. The HVS is also more acute to luminous intensity than chromatic intensity. Therefore, different quantizer matrices are developed for each. An example of a luminance quantization matrix widely used in JPEG compression is given in Figure II-6 [4]. The dimensions match the 8×8 block size used with the 2-D DCT. In application of a quantizer matrix, the entire matrix can be scaled by a multiplicative constant in order to scale quantization noise and compression while maintaining the relative importance among the coefficients.

$$\begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

Figure II-6: HVS-Based Luminance Quantization Matrix [4].

2. Bit Allocation

Using a bit allocation approach, the value of the step size is chosen to minimize distortion within a bit budget. It is a classical resource allocation problem, where the fundamental trade-off in quantization is between rate (number of bits) and distortion (approximation error) formalized as rate-distortion theory. Several sophisticated algorithms utilize Lagrange methods applied to arbitrary rate-distortion curves. Of these, a popular approach is varying the step size in proportion to the variance of the coefficient

[17]. However, bit allocation schemes, regardless of the methodology, do not account for the sensitivity to different spatial frequency characteristics of human visual perception.

D. ENTROPY ENCODING

Further compression can be realized through a lossless process called entropy encoding that removes redundant information. The simplest style of entropy encoding is run-length encoding (RLE). With RLE, a data set is parsed to locate sequences of repeated values. Any such sequence is replaced by a codeword consisting of a delimiter, the value, and the number of times the value is repeated. The longer the sequence of the repeated value, the greater the available compression. Following quantization, many elements of a coefficient block typically have a value of zero. Often many high frequency rows and columns are completely composed of zeros. Accordingly, it is advantageous to scan the coefficient block in a manner as to produce the longest runs of zeros. In JPEG compression, scanning the quantized coefficient block as a vector in zigzag fashion starting with the DC coefficient down to the $F_{8,8}$ coefficient has been shown to increase the run-length of zeros [4]. Since the repeated value here is known (0), an adaptation of the basic codeword scheme cited above consists of the run-length of zeros followed by the magnitude of the next non-zero value. If there is no remaining non-zero value in the block, a special end-of-block (EOB) codeword is used instead.

After RLE, the quantized coefficient block is represented by a set of codewords, where each codeword represents a symbol drawn from a larger source alphabet. Variable-length coding (VLC) minimizes the average codeword length by assigning shorter codewords to the most probable symbols and longer codewords to the least likely occurring symbols while maintaining each uniquely decipherable (UD). Huffman coding is the most widely used entropy-encoding algorithm and is guaranteed to produce a minimum average length, UD code [15]. The Huffman algorithm uses each symbol's probability of occurrence and builds a prefix code using an optimum binary-branching tree. Since both the coder and decoder need to use the same coder and generating a Huffman table is computationally expensive, standard tables are normally pre-defined

using data drawn from test images. Optimal compression is no longer guaranteed, but encoding and decoding are faster, and the need to transmit the VLC table is avoided.

E. FRAME CODING

As discussed previously, intraframe coding compresses each frame of a video as a separate, still image. Its advantage is superior error resilience; its disadvantage is limited compression gain. Interframe coding exploits frame-to-frame correlation by coding only the difference in successive frames, thereby allowing the potential for higher compression gain than that achieved by intraframe coding alone. The disadvantage to interframe coding is in robustness since decode errors can propagate between frames and spatially within a frame depending on the algorithm.

Because of the compression required for the target VTC scenario, strict intraframe coding is not an option. Therefore, a robust algorithm for interframe coding is needed. Several source-coding techniques used to exploit temporal redundancy, such as motion compensation, differential pulse code modulation (DPCM) and block updating, are discussed in [4]. For a robust application in a real-time, heterogeneous, multicast environment, block updating was judged the most promising and is discussed further.

Interframe coding via block[4] updating is a variation on intraframe coding. With block updating, each block in the current frame is compared to the corresponding block in the previous frame, and a distance metric is calculated. If the metric is above a threshold value, that block is intracoded and transmitted. Otherwise, no further consideration is given to that block; it is skipped. Thus, block updating conserves bandwidth by coding and transmitting only those blocks that have changed perceptually since the previous frame [18]. In low-motion video, such as the "talking head" scenario, motion is confined to a relatively small region within a frame and the background remains static. A significant bandwidth savings is possible.

[4] In this context, "block" implies an $N \times N$ pixel region, not necessarily 8×8.

In order for block updating to be effective, a suitable distance metric must be defined. Common distance metrics employed are MSE, sum of absolute differences (SAD), and absolute sum of differences (ASD) [4] [14].

Utilizing block updating alone for a tactical VTC, where end-users may join a session that is already in progress, has a liability. End-users joining late will never receive a block that does not exceed the selection threshold, thus leaving them with a partially reconstructed video. In order to mitigate this affect, block updating can be combined with an aging algorithm that periodically forces block updates. Such an algorithm guarantees a full scene reception within some set interval [14].

F. MEASURING QUALITY

Given that video coders trade compression gain for image quality, quantifying the level of distortion introduced is useful in evaluating different coding schemes. A useful measure of image distortion D is the MSE between the original (x) and reconstructed ($\hat{x}$) images [4]:

$$D = \frac{1}{MN}\sum_{n=1}^{N}\sum_{m=1}^{M}\left(x_{n,m} - \hat{x}_{n,m}\right)^2 \tag{II-7}$$

Using distortion D, the signal-to-noise ratio (SNR) is determined as

$$SNR = 10\log_{10}\frac{\sigma^2}{D}, \tag{II-8}$$

where σ^2 is the variance of the original image. The most widely published measure of image quality is the peak signal-to-noise ratio given by

$$pSNR = 10\log_{10}\frac{K^2}{D}, \tag{II-9}$$

where K is the maximum peak-to-peak value in the image, 255 for the typical 8-bit image. For example, a typical peak SNR for a typical JPEG encoded grayscale image is 28 dB at 0.5 bpp.

Using MSE as a measure of image quality does have drawbacks. MSE does not directly define perceptual quality since all errors are given equal weight. Two

compression techniques yielding the same MSE for an image may deliver slight differences in perceptual quality.

III. LAYERED VIDEO CODECS

A. LAYERED CODING SCHEMES

Parker [9] discusses several previous approaches to layered video coding. They can be classified into three fundamental categories: component-based layering, spatial-based layering, and frequency-based layering.

Component-based layered coders transmit a base layer and usually a single enhancement layer. A traditional DCT-based coder (yielding some set quality) supplies the base layer. The enhancement layer improves this quality by either providing augmenting information to the base layer or correcting distortion present within the base layer. Two such schemes are proposed by Rhee and Gibson [8] [20]. Implicit in the component-based approach is that the enhancement layer may duplicate information already present in the base layer.

Spatial-based layered coders partition a frame into hierarchical areas of interest and encode each area separately. For example, the "talking head" frame shown in Figure III-1 may be partitioned into two areas of interest with the speaker being of primary interest and the remainder of the frame being of secondary interest. With spatial-based layering, relatively more bandwidth would be allocated to the region encompassing the speaker with the balance of available bandwidth being apportioned to the rest of the frame. Bahl and Hsu [21] have proposed a coder that incorporates content-sensitive spatial decomposition and multiresolution coding. The difficulty with implementing a spatial-based coder is creating the hierarchical areas of interest dynamically while minimizing the overhead required to identify their shifts relative to the layer assignments.

Figure III-1: Sample "Talking Head" Video Frame.

Frequency-based layered coders decompose a frame into subbands and then arrange the subbands into individual layers. The frequency decomposition may be applied to the entire frame or individual macroblocks within the frame. Each layer may contain one or more subbands. The base layer contains at least the low frequency subband but may also contain higher frequency subbands in order to improve base layer quality. Figure III-2 illustrates this concept using the DWT to decompose a frame, where the LL subband alone constitutes the base layer, the LH and HL subbands are combined into a first enhancement layer, and the HH subband constitutes the second enhancement layer.

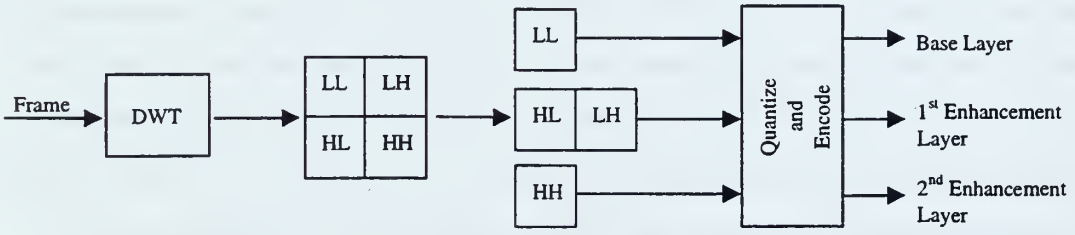

Figure III-2: Basic Layered Video Coder Using the DWT.

The principle advantages to frequency-based layering are its extendibility and flexibility. Further decomposing the frame through application of the DWT to each subband depicted in Figure III-2 creates a total of 16 subbands. Repeated application of the DWT to these subbands creates even more subbands. With a larger number of subbands, there is increased flexibility in the manner in which they can be assigned to

layers. This, in turn, facilitates exploiting each subband's perceptual importance. Additionally, a greater number of subbands permits a greater possible number of layers. Three coders that utilize the frequency-based approach are discussed in [14], [22], and [23].

B. A LOW-COMPLEXITY, ADAPTIVE CODER FOR TACTICAL VTC

As cited above, several diverse approaches to designing layered coders have been proposed; each emphasizes different network architectures or applications to varying degrees. However, there is neither a consensus in identifying a preferred, structured approach nor a consensus in quantifying those parameters that make a layered coder *effective*. This section presents the development and implementation of a new layered coder that is specifically tailored for the tactical VTC scenario where limited transmission bandwidth is available, and the robustness of transmission is at a premium.

Both the characteristics of the tactical VTC application and the desire to quantify effectiveness guide the strategy for developing and implementing the new layered coder. Specifically, the application yields five requirements. The coder must 1) provide a video stream characterized by the bandwidth, resolution, frame rate, and color depth detailed in Table I-1, 2) optimize compression adaptively for both low motion video and static slides, 3) provide a low complexity architecture to minimize coding delays and power requirements, 4) provide error resilient decoding at high packet loss rates, and 5) constrain bit rate to a predetermined average.

The guiding factors stemming from the desire to quantify effectiveness are twofold. First, the coder must provide a base layer with acceptable quality and two (or more) enhancement layers that progressively improve *perceptual* quality. Second, the coder must minimize the bitstream overhead required to accommodate the layering structure. A functional diagram of the implemented coder is shown in Figure III-3. Each component is addressed in follow on sections.

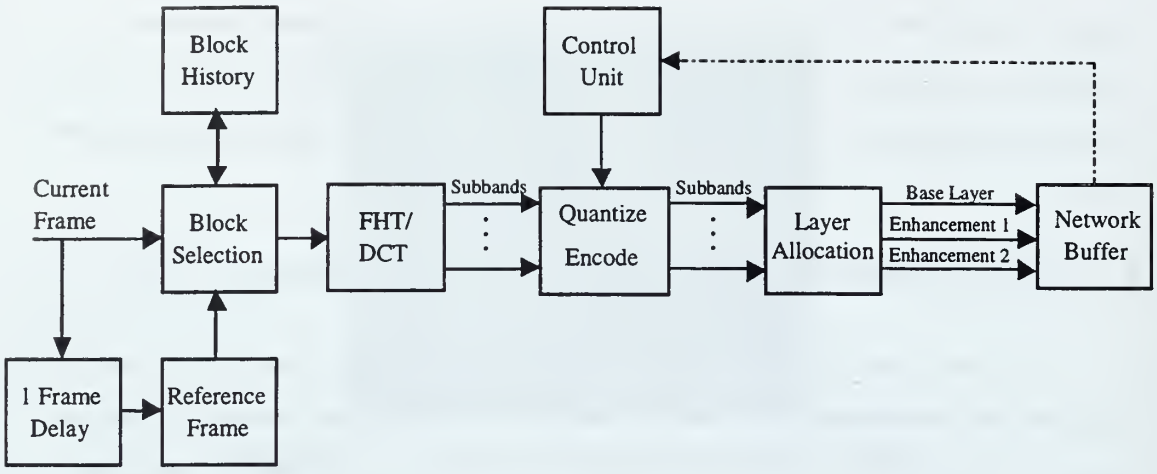

Figure III-3: Functional Block Diagram of Proposed Layer Coder.

1. Temporal Compression via Block Updating

Temporal compression is accomplished via block updating applied at the macroblock level. Only those macroblocks exhibiting sufficient change on a frame-to-frame basis are encoded. Although block updating has been shown to yield inferior compression performance relative to motion prediction algorithms, block updating was chosen because the performance differential is small when low activity video is considered [24], and greater robustness is realized because temporal error propagation is greatly limited and spatial error propagation is eliminated. Block updating also negates the need for the locally decoded reference frame that is common to motion prediction schemes. This greatly simplifies the coder by eliminating the overhead required by the performance of an inverse quantization/transform at the coder. Block selection, as considered here, is solely with regard to motion video sequences. Since static slide sequences exhibit little or no motion, block selection via motion detection is of limited utility there. Indeed, transmissions during static sequences result from considerations presented in the next section, where macroblock aging is discussed.

Sufficient change between corresponding, frame-successive macroblocks is determined by calculating a distance metric between them and comparing the result to a threshold. The distance metric utilized is the non-normalized ASD given by:

$$ASD = \left| \sum_{m=1}^{M} \sum_{n=1}^{N} \left(x_{m,n} - x_{m,n}^{R} \right) \right|, \qquad \text{(III-1)}$$

where $x_{n,m}$ represents the pixel value within the current block while $x_{m,n}^{R}$ represents the pixel value in the reference block. The primary reasons for choosing ASD over other metrics are twofold. First, the ASD is computationally efficient compared to MSE and SAD. The ASD employs only additions and subtractions and a single absolute value operation. MSE requires expensive multiplication operations – making it ill suited to real-time applications. SAD requires the same number of additions and subtractions as ASD, but it requires $M \times N - 1$ more absolute value operations. Second, since the ASD takes a single absolute value of a sum, it acts as an accumulator and provides a lowpass filtering effect that removes noise in pixel intensities introduced through video capture. This smoothing prevents the threshold from being exceeded spuriously in otherwise static regions of the frame sequence whereas the nonlinear operations performed on a per-pixel basis in MSE and SAD tend to accumulate this noise energy and yield spurious block selections. Thus, the ASD metric allows bandwidth to be more effectively devoted to regions of interest [14].

Two independent elements affect video quality and, therefore, the required bit rate: adequate motion detection (to prevent "jerky" reconstructed video) and control of distortion introduced through quantization. The goal in motion detection is to select the *maximum* block selection threshold that adequately captures motion. In the video sequences examined, a threshold of 160 (for ASD) proved adequate for detecting perceptual motion and resulted in an average of 24.8 macroblocks selected per frame.

Further considering the issue of computational expense, the ASD metric is applied to individual 8×8 blocks within the macroblock; the first block to exceed the threshold triggers macroblock selection and ends the search. This avoids the expense of examining the remaining blocks of the macroblock. Additionally, since the HVS acuity is more sensitive to changes in luminous than chromatic intensity [4], distance calculations are confined to the luminous component of each pixel even if chromatic information were available.

active content (though still considered low-motion video) of all the test video sequences examined. For this sequence, the average number of macroblocks selected per frame was 34.61 and their standard deviation was 10.19. The threshold was set at three standard deviations above the mean (65). The comparison operation is performed in the main code block, *thesis.m*. A frame sequence is determined to be static when macroblocks chosen for transmission result solely from aging. In this case, the fixed quantizer triplet and the static VLC table from Appendix B are used. Otherwise, the sequence is deemed motion video, and the bit rate control scheme discussed above is employed with its associated custom VLC table from Appendix B. This determination is performed within the function, *m_blk_id_xr.m*.

This chapter began with a presentation of the three basic techniques available for layered video coding. The approach implemented here, as proposed by Parker [9], is frequency-based and utilizes the FHT and the 2-D DCT for motion video sequences and the FHT alone for static slide sequences. Different frequency transforms are used due to the inherent differences in the perceptual frequency content within the two types of sequences. The method of implementing frame refreshment was detailed as a block selection scheme applied at the macroblock level that captures perceptual changes due to motion within a scene and limits the duration of decoder errors at receivers by forcing macroblock updates via an aging algorithm. Quantization and encoding techniques were presented with the implementation including the use of the JPEG standard and uniform quantization coupled with one of two custom VLC tables. The issue of rate control was addressed, and the implementation of a scheme that reduces a 4-D rate control surface to a simple quantizer table lookup to control bit rate at the frame level was discussed. Finally, the manner by which the coder detects a scene change and determines the type of sequence under consideration was delineated.

IV. RESULTS

This chapter presents some results from a short video segment consisting of 100 frames of a single speaker followed by 50 frames of a presentation slide filled with line diagrams and text. The Matlab code is contained in Appendix C. A sample frame from each sequence is shown in Figure IV-1 and Figure IV-2. Each shows the original frame and the reconstructed frame with only the base layer received, with the base layer and the first enhancement layer received, and with all layers received.

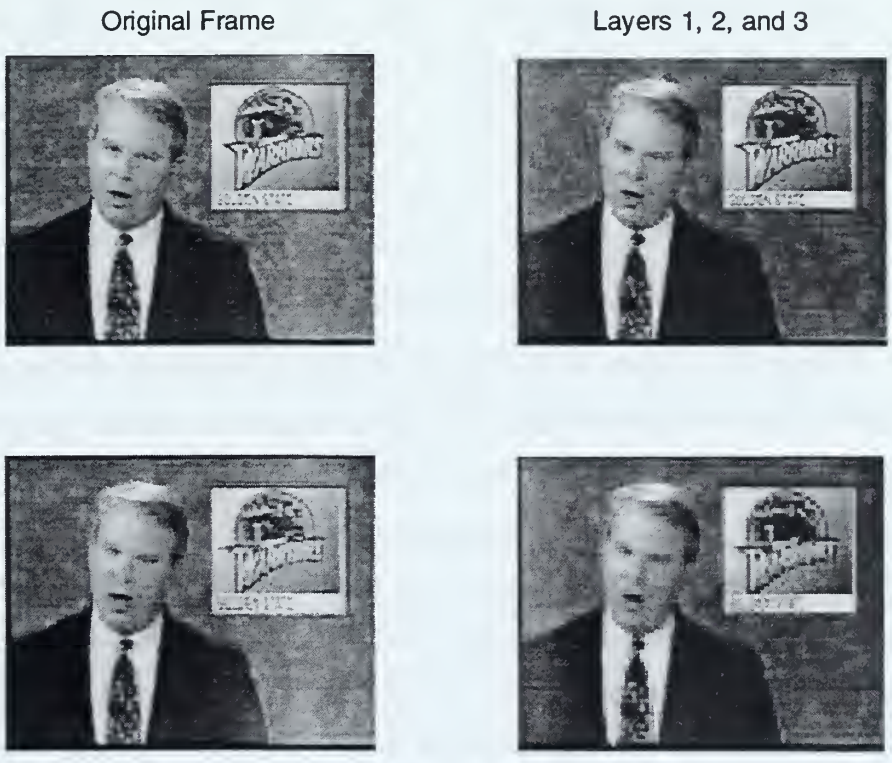

Figure IV-1: Original and Reconstructed Frames from a Motion Video Sequence.

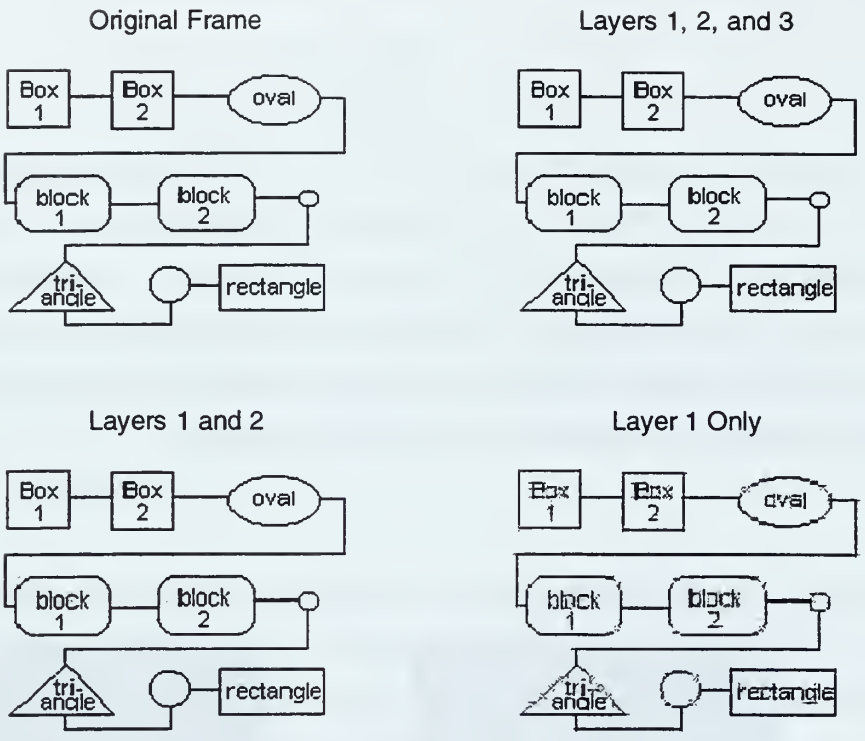

Figure IV-2: Original and reconstructed Frames from a Static Video Sequence.

Figure IV-3 shows the bit rate traces for the 150-frame layered video segment. Part (a) displays the traces resulting from the constant motion video quantizer triplet ({6,12,20}) expected to yield 80 kbps based on the average bit rate resulting form all test motion video sequences and the fixed static sequence quantizer triplet. Part (b) displays the traces resulting from the bit rate control scheme for the motion video frames with the coder attempting to achieve an average 80 kbps and the fixed static sequence quantizer triplet. Bit rate spike suppression is employed for the initial frame of each scene as discussed previously. The distribution of bit rate offered by a layered video coder is evident with the bit rate ratio among layers being approximately 5:3:2 for both sequences. As congestion occurs in the network, the higher layers can be dropped to combat the congestion while maintaining much of the quality as illustrated in Figure IV-1 and Figure IV-2. Neglecting the initial frame, the average bits per frame for the motion video

sequence without the control scheme is 7454 bpf with a standard deviation of 1362 bpf. With the control scheme, the average and standard deviation are 7988 bpf and 942 bpf, respectively. As expected, the bit rate from the static sequence is much lower since the bit rate results solely form macroblock aging. This illustrates that rate control is not of significant benefit for static sequences.

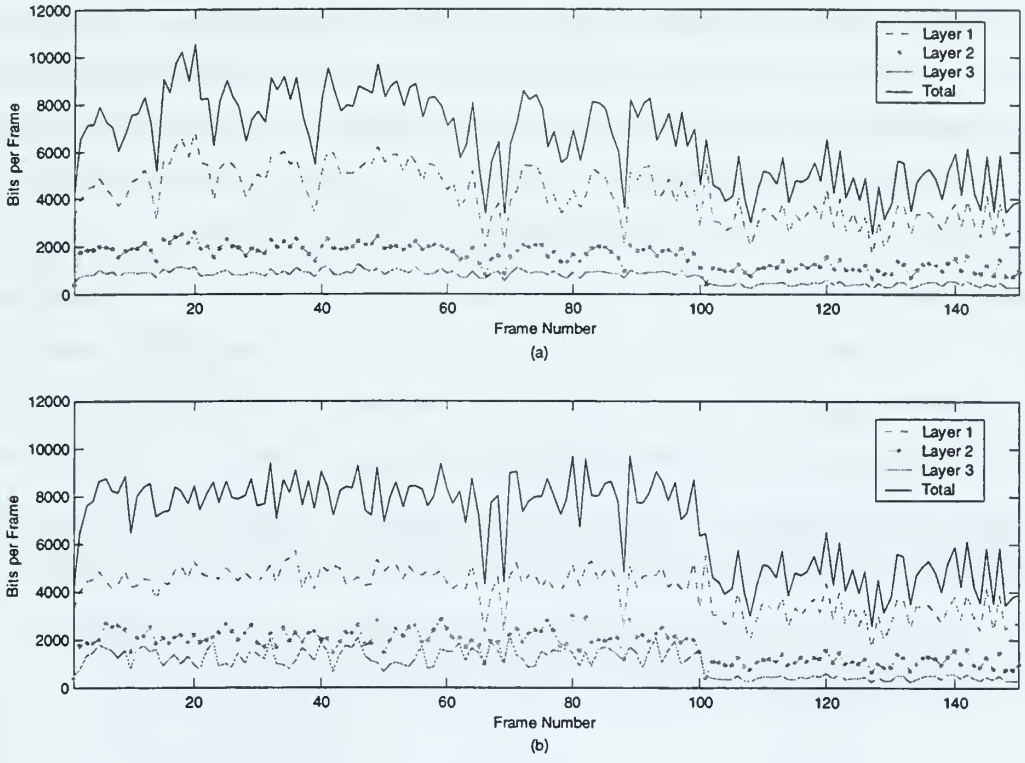

Figure IV-3: Bit Rates for (a) Fixed Quantization and (b) Bit Rate Control.

Figure IV-4 quantifies the progressive improvement in quality of the reconstructed video based on pSNR for the same 150-frame layered video segment illustrated in Figure IV-1 and Figure IV-2. At the beginning of each sequence, quality ramps up over the aging interval following a scene change. After this period, quality is observed to remain relatively flat for each sequence regardless of the number of layers. For the motion video sequence, the base layer provides a smoothed but acceptable display. Text is not readable but the speaker's movements are easy to follow. Adding the first enhancement layer improves sharpness and adds a 4 dB improvement in pSNR

although small text is still difficult to discern. The second enhancement layer only adds 1-2 dB improvement, but small text is clearly readable and other features with fine edges are sharper. With static video, the role of the enhancement layers is even more dramatic. Even though most of the macroblock's energy is included in the base layer and contributions from each frequency band are included, the base layer still shows much softness although the shapes are readily identifiable. The first enhancement layer adds a 7 dB improvement and dramatically improves sharpness. The final layer, even though the bit rate contribution is the smallest of the three layers, almost doubles the pSNR, and the reconstructed frame is virtually identical to the original frame. Neglecting the initial frame, the average pSNR utilizing all layers for the motion video sequence without the control scheme is 29.5 dB with a standard deviation of 1.7 dB. With the control scheme, the average and standard deviation are 29.8 dB and 1.9 dB, respectively. Note that these values include the ramp up in quality following the initial scene change. These average pSNR values are not directly comparable, however, because of the difference in average bit rates; the higher quality, rate control approach uses approximately 500 more bits per frame. But since this quality is achieved within the desired bit budget of 80 kbps, the rate controller allows more effective utilization of the available bandwidth to better image fidelity. The statistics obtained from the motion video sequence traces in Figure IV-3 and Figure IV-4 are summarized in Table IV-1.

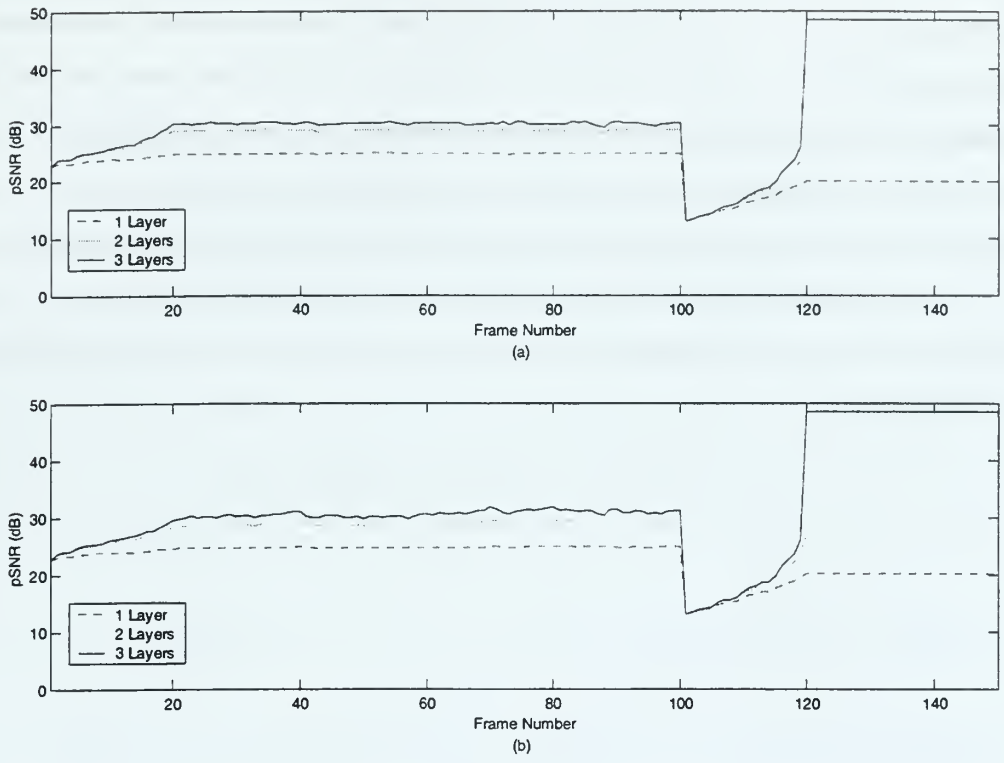

Figure IV-4: pSNR for (a) Fixed Quantization and (b) Bit Rate Control.

Parameter	With Rate Control	Without Rate Control
Mean Bit Rate (bpf)	7998	7454
Bit Rate STD (bpf)	942	1362
Mean pSNR (dB)	29.83	29.51
pSNR STD (dB)	1.92	1.74

Table IV-1: Rate Controlled and Uncontrolled Motion Video Sequence Statistics.

Additionally, the issue of the layered video codec's resilience to bit errors introduced during transmission was examined using the same motion video sequence as above. Using loss rates of 10%, 25%, and 50%, four different case were tested. Case one distributed the bit errors across the layers in proportion to their contribution to the total bit rate and utilized zero-order error concealment; the reconstructed frame retained the content of the previous frame for that portion lost during transmission. If the loss

occurred in the base layer, the enhancement layers were neglected. If the loss occurred in an enhancement layer, the reconstruction was performed utilizing the base layer and the other enhancement layer. Case two treated all layers as a single video stream and utilized zero-order error concealment; that is, a transmission loss was a loss for all layers. Cases three and four are identical to cases one and two, respectively, except that no error concealment was used. Instead an information loss caused the decoder to assign the value of zero to all coefficients in the affected layers. As Figure IV-5 illustrates, spreading bit errors across multiple layers has less negative impact on the reconstructed image at high loss rates.

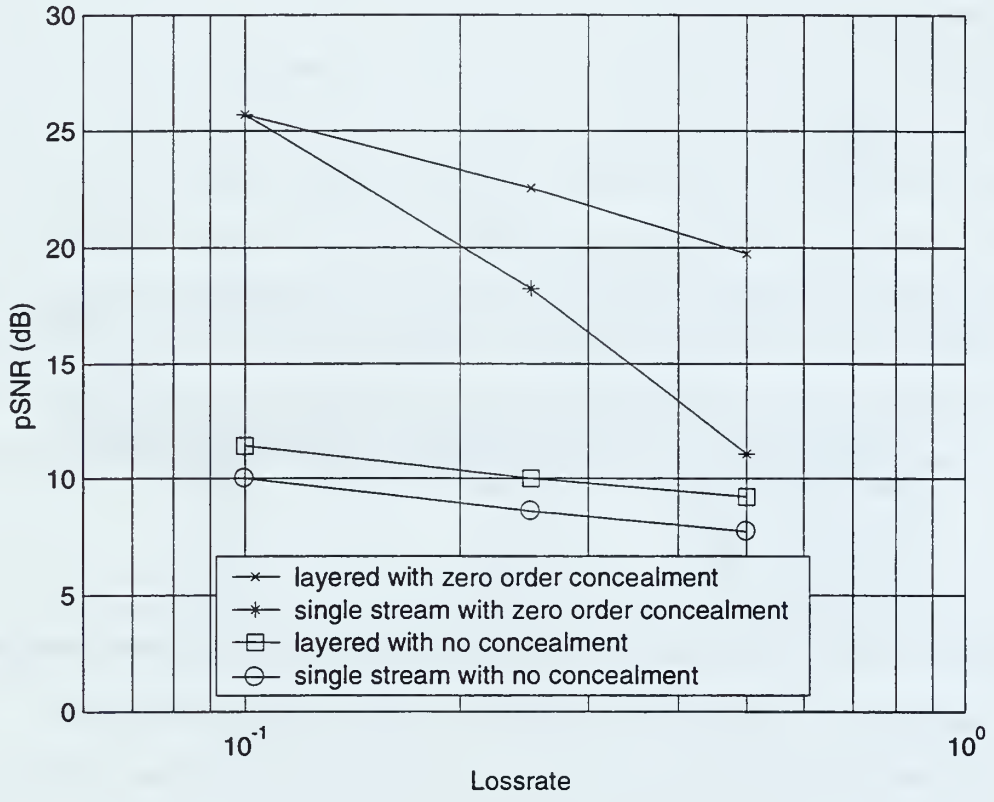

Figure IV-5: Comparison of Error Resilience.

Considering static slide sequences, the relationship is more complicated. In general, since the frame content is entirely static, the best reconstruction in an error prone environment is to forgo a macroblock update if the update would be made with fewer

layers than the present reconstruction of that macroblock. Further study into the implementation of such a scheme is warranted.

This chapter presented representative frames from a video segment consisting of motion video and static slides. For each type of content, the original frame and the reconstructions with one, two, and three layers were given. Also presented were plots of the bits per frame and pSNR as a function of the frame sequence for both fixed quantization and variable quantization using the bit rate control scheme. These plots served to quantify the quality depicted in the reconstructions and to illustrate the bit allocation among the layers. Finally, the effect of spreading bit errors across layers compared to confining them to a single video stream was presented.

V. CONCLUDING REMARKS

A. CONCLUSIONS

This thesis has presented the development and implementation of a new layered video codec proposed by Parker [9] that emphasizes robust transmission of a video teleconference (VTC) at low bit rates. The dual nature of the targeted scene content – low-motion video and static slide sequences consisting of text and line drawings – and the assumed application in a multicast, heterogeneous, wireless network environment required significant flexibility in the implementation. The essential features of the coder are summarized as follows.

Frame refreshment is accomplished via block updating and an aging algorithm, both applied at the macroblock level. This approach promotes greater robustness because spatial error propagation is eliminated and temporal error propagation is greatly limited. The combined technique captures perceptual changes due to motion within the scene, limits the duration of error artifacts in the reconstruction at receivers, and ensures that new participants in a VTC session that is already in progress receive a complete frame in a timely manner.

The macroblocks selected for transmission are decomposed in frequency using the fast Haar transform (FHT). For motion video sequences, the lowpass subband is further processed with the two-dimensional discreet cosine tranform. The horizontal and vertical edge detail subbands are further decomposed with a repeated application of the FHT. Static slide sequences are decomposed solely by a second order FHT analysis.

The lowpass subband of motion video is quantized and encoded using the JPEG standard in order to exploit the human visual system perceptive characteristics. The remaining subbands of motion video sequences are subjected to uniform quantization and encoding with a custom variable length coding (VLC) table. All subbands of static slides are subjected to uniform quantization and encoding with a separate custom VLC table. All quantization is performed using a triplet of quantizers, and each subband is quantized with one of the triplet parameters based on the variance of subband coefficients.

Subbands are assigned to layers by grouping bands of coefficients with similar variances into a common layer. Three layers are used in the coder. The base layer is independently decodable and yields an acceptable, minimum-quality reconstruction; each enhancement layer progressively improves the quality of the reconstruction.

Bit rate is controlled at the frame level by selecting the quantizer triplet to be used in the current frame based on the number of bits used in the previous frame, the desired average bit rate, and the number of macroblocks selected for transmission. The implementation involves a simple table lookup, which resulted from the optimal one-dimensional reduction of a four-dimensional control surface.

B. FUTURE WORK

The implementation and the results presented here suggest that the layered video codec has potential practical utility to video teleconferencing in multicast, heterogeneous, wireless networks. Now, several aspects of the coder can be pursued further. For example, although three layers were used in the present implementation, the techniques employed can be used to scale the coder to include an arbitrary number of layers. Work on techniques to dynamically change the layering scheme within a sequence is desirable. The ability to handle color and audio needs to be incorporated into the code. Further refinement of the block search pattern utilized for macroblock selection and the possibility of rate control at the macroblock level can be evaluated. Also, with regard to static slide sequences, investigation of the ability to detect and reconstruct slight movement within such a frame, such as the movement of a cursor, is warranted. Finally, implementation in a high level language or hardware is another potential future task.

APPENDIX A. MOTION VIDEO VARIABLE LENGTH CODİNG TABLE (VLC)

The following is the custom VLC table used with motion video sequences. The last character in the codeword *s* indicates the appended sign bit.

INDEX	LAST	RUN	LEVEL	BITS	CODEWORD
1	0	0	1	3	11s
2	0	0	2	5	0101s
3	0	0	3	6	10010s
4	0	0	4	9	101000011s
5	0	0	5	12	101110011101s
6	0	0	6	14	10111111001110s
7	0	1	1	5	0100s
8	0	1	2	7	1001111s
9	0	1	3	10	0010001015
10	0	1	4	12	101111110000s
11	0	1	5	17	10110100100000111s
12	0	1	6	14	10110100101005
13	0	2	1	5	0000s
14	0	2	2	8	10110015
15	0	2	3	12	011100010105
16	0	2	4	13	0010001101105
17	0	3	1	6	01111s
18	0	3	2	9	011001005
19	0	3	3	11	10111001015
20	0	3	4	13	1011010110015
21	0	4	1	6	00011s
22	0	4	2	9	011100105
23	0	4	3	10	1011010105
24	0	4	4	15	001011010011115
25	0	5	1	6	10010s
26	0	5	2	10	0001011015
27	0	5	3	12	101111110005
28	0	5	4	16	1011010010001115
29	0	6	1	7	0111015
30	0	6	2	10	0001011105
31	0	6	3	14	01110001000005
32	0	6	4	16	1011010010001105
33	0	7	1	7	011011s
34	0	7	2	10	0010001005
35	0	7	3	14	10111111001115
36	0	7	4	16	1011010010000015
37	0	8	1	7	011000s
38	0	8	2	9	001001005
39	0	8	3	10	0010110115
40	0	8	4	13	0111000100015
41	0	9	1	8	00101115
42	0	9	2	11	001001101005
43	0	9	3	14	10011010110005
44	0	10	1	8	10110115
45	0	10	2	11	10011010015
46	0	10	3	14	00100110110105
47	0	10	5	16	1011010010000005

48	0	11	1	9	0 0 1 0 1 1 0 0 s
49	0	11	2	12	0 0 1 0 0 0 1 1 1 0 0 s
50	0	11	3	14	1 0 1 1 0 1 0 0 1 0 1 0 1 s
51	0	12	1	8	0 0 1 0 0 0 0 s
52	0	12	2	10	0 1 1 1 0 0 0 1 1 s
53	0	12	3	13	1 0 1 1 1 1 1 0 0 1 0 0 s
54	0	12	4	15	0 0 0 1 0 1 1 0 0 1 1 1 0 1 s
55	0	13	1	9	1 0 1 1 1 0 0 0 s
56	0	13	2	12	1 0 1 1 1 1 1 1 1 1 1 s
57	0	13	3	15	0 0 1 0 1 1 0 1 0 0 1 1 0 0 s
58	0	14	1	9	0 0 1 0 0 1 0 1 s
59	0	14	2	13	0 0 1 0 0 0 1 1 0 1 1 1 s
60	0	14	3	17	1 0 1 1 0 1 0 0 1 0 0 1 1 0 1 0 s
61	0	15	1	10	0 1 1 0 0 1 0 1 0 s
62	0	15	2	14	1 0 1 1 0 1 0 0 0 1 0 1 1 s
63	0	16	1	10	1 0 1 0 0 0 1 0 0 s
64	0	16	2	15	1 0 0 1 1 0 1 1 1 1 0 0 1 0 s
65	0	17	1	11	0 0 0 1 0 1 1 1 1 0 s
66	0	17	2	17	0 1 1 0 0 1 0 1 1 0 0 1 1 1 1 1 s
67	0	18	1	11	1 0 1 0 0 0 1 0 1 1 s
68	0	19	1	11	1 0 1 0 0 0 1 0 1 0 s
69	0	20	1	11	1 0 1 1 0 1 0 0 1 1 s
70	0	20	2	17	0 1 1 0 0 1 0 1 1 0 0 1 1 1 1 0 s
71	0	21	1	12	0 1 1 0 0 1 0 1 1 0 1 s
72	0	22	1	12	0 1 1 1 0 0 0 1 0 1 1 s
73	0	22	2	16	1 0 1 1 0 1 0 0 1 0 1 1 1 0 1 s
74	0	23	1	12	0 0 0 1 0 1 1 1 1 1 1 s
75	0	23	2	15	0 0 1 0 1 1 0 1 0 0 1 1 0 1 s
76	0	24	1	11	0 0 1 0 0 0 1 1 0 0 s
77	0	24	2	16	1 0 1 1 0 1 0 0 1 0 1 1 1 0 0 s
78	0	24	3	17	1 0 1 1 0 1 0 0 1 0 0 0 0 1 0 0 s
79	0	25	1	12	0 0 1 0 0 1 1 0 0 1 1 s
80	0	25	2	16	1 0 1 1 0 1 0 0 1 0 1 1 1 1 1 s
81	0	26	1	12	1 0 1 1 1 1 1 1 0 0 1 s
82	0	26	2	17	1 0 1 1 0 1 0 0 0 0 1 0 1 0 0 0 s
83	0	27	1	12	0 0 0 1 0 1 1 0 0 0 0 s
84	0	27	2	17	1 0 1 1 0 1 0 0 0 0 1 0 1 0 1 1 s
85	0	28	1	12	0 0 1 0 0 0 0 1 1 1 0 s
86	0	28	2	17	1 0 1 1 0 1 0 0 0 0 1 0 1 0 1 0 s
87	0	29	1	12	1 0 1 1 1 1 1 1 0 0 0 s
88	0	29	2	17	1 0 1 1 0 1 0 0 1 0 0 1 0 1 0 1 s
89	0	30	1	12	1 0 1 1 0 1 0 1 1 1 0 s
90	0	30	2	15	1 0 1 1 0 1 0 0 0 0 1 0 0 0 s
91	0	31	1	12	1 0 0 1 1 0 1 1 1 0 0 s
92	0	31	2	17	1 0 1 1 0 1 0 0 1 0 0 1 0 1 0 0 s
93	0	32	1	12	1 0 0 1 1 0 1 0 1 0 1 s
94	0	32	2	14	1 0 1 1 0 1 0 0 0 1 0 1 0 s
95	0	33	1	13	1 0 1 1 1 1 1 1 1 1 0 0 s
96	0	33	2	17	1 0 1 1 0 1 0 0 1 0 0 1 0 0 0 1 s
97	0	34	1	13	0 0 1 0 0 0 1 1 1 1 1 0 s
98	0	34	2	17	1 0 1 1 0 1 0 0 1 0 0 1 0 0 0 0 s
99	0	35	1	13	1 0 1 1 1 1 1 1 0 1 0 0 s
100	0	35	2	17	1 0 1 1 0 1 0 0 1 0 0 1 0 0 1 1 s
101	0	36	1	12	1 0 0 1 1 0 1 0 1 0 0 s
102	0	36	2	17	1 0 1 1 0 1 0 0 1 0 0 1 0 0 1 0 s
103	0	37	1	13	0 0 0 1 0 1 1 1 1 1 0 0 s

104	0	38	1	12	101110011110s
105	0	39	1	12	101101000011s
106	0	40	1	13	1011111111001s
107	0	40	2	16	0010110100010100s
108	0	41	1	13	1011100011110s
109	0	41	2	17	10110100100011101s
110	0	42	1	14	01110001001100s
111	0	43	1	13	0110010110000s
112	0	43	2	17	10110100100011100s
113	0	44	1	13	1011100010001s
114	0	45	1	12	101101011101s
115	0	46	1	14	01110001001111s
116	0	46	2	16	1011010010111110s
117	0	47	1	13	1001101111011s
118	0	48	1	12	100110100000s
119	0	48	2	15	000101100100000s
120	0	49	1	13	0110010111111s
121	0	50	1	13	1011010000011s
122	0	51	1	14	10111001100010s
123	0	52	1	14	01110001001110s
124	0	53	1	14	10011010111100s
125	0	53	2	16	1011010010111001s
126	0	54	1	14	10111111001100s
127	0	54	2	16	0010110100101111s
128	0	55	1	14	0001011001101s
129	0	56	1	13	1011111111000s
130	0	56	2	15	000101100100011s
131	0	57	1	14	00010110011100s
132	0	57	2	16	0010110100100001s
133	0	58	1	15	1001101111100011s
134	0	59	1	16	0110010110011110s
135	0	60	1	16	0001011001111000s
136	0	61	1	17	00010110011110010s
137	0	62	1	15	0010110100000 0s
138	1	0	1	5	0011s
139	1	0	2	7	011010s
140	1	0	3	8	10110000s
141	1	0	4	11	00100110000s
142	1	0	5	15	0010110100100 01s
143	1	1	1	6	10001s
144	1	1	2	10	1011111101s
145	1	1	3	13	1011111110101s
146	1	1	4	15	1011010000101 1s
147	1	1	5	17	10110100100000110s
148	1	2	1	7	001010s
149	1	2	2	11	00101101 01s
150	1	2	3	13	1001101111101s
151	1	2	4	16	1011010010000101s
152	1	3	1	6	10101s
153	1	3	2	10	1001101110s
154	1	3	3	13	0010011001100s
155	1	3	4	16	0010110100100000s
156	1	4	1	7	101001s
157	1	4	2	12	001001101111s
158	1	4	3	16	1011010010111000s
159	1	4	4	16	1011010010000100s

160	1	5	1	8	1011110s
161	1	5	2	15	000101100010001s
162	1	5	3	15	000101100010010s
163	1	6	1	8	0001001s
164	1	6	2	13	1001101011101s
165	1	7	1	8	0110011s
166	1	7	2	12	100110111111s
167	1	7	3	14	10011010111001s
168	1	8	1	8	0001010s
169	1	8	2	13	1011100100010s
170	1	8	3	16	1011010010110111s
171	1	9	1	9	011100111s
172	1	9	2	15	001011010000011s
173	1	9	3	17	10110100100011000s
174	1	10	1	8	1001100s
175	1	10	2	15	001011010000010s
176	1	11	1	8	1010000s
177	1	11	2	14	10011010111110s
178	1	11	3	17	10110100100011011s
179	1	12	1	8	1011101s
180	1	12	2	13	0010111010001s
181	1	12	3	16	1011010010110110s
182	1	13	1	9	001001111s
183	1	13	2	14	01111000100001s
184	1	14	1	9	011100000s
185	1	14	2	18	000101100111001111s
186	1	14	3	17	10110100100000101s
187	1	15	1	8	0001000s
188	1	15	2	13	0110010111110s
189	1	15	3	14	00010111111011s
190	1	15	4	15	001011010011110s
191	1	16	1	12	101101011111s
192	1	16	2	16	0010110100101101s
193	1	17	1	13	0110010111100s
194	1	18	1	14	00010111111010s
195	1	19	1	13	0010001101101s
196	1	20	1	13	0010011011100s
197	1	20	2	18	000101100111001110s
198	1	21	1	14	10111111111011s
199	1	22	1	13	0010001111111s
200	1	23	1	12	000101100001s
201	1	23	2	15	101101000001001s
202	1	24	1	14	01100101100010s
203	1	24	2	17	10110100000101001s
204	1	25	1	13	1011010110000s
205	1	26	1	14	01100101110011s
206	1	27	1	14	10111001100011s
207	1	28	1	14	10110100000011s
208	1	29	1	13	1011010000000s
209	1	30	1	13	1001101000011s
210	1	31	1	13	1011100011000s
211	1	31	2	17	10110100100010111s
212	1	32	1	14	10111111111010s
213	1	32	2	17	10110100100010110s
214	1	33	1	15	011001011001110s
215	1	34	1	14	01100101110110s

216	1	35	1	14	0 1 1 1 0 0 0 1 0 0 1 0 1 s
217	1	36	1	14	0 0 1 0 0 0 1 1 1 0 1 0 1 s
218	1	37	1	14	0 0 1 0 0 1 1 0 1 1 0 1 1 s
219	1	38	1	13	1 0 1 1 1 0 0 1 1 1 1 1 s
220	1	39	1	13	0 0 1 0 0 1 1 0 0 1 0 1 s
221	1	40	1	14	0 0 1 0 0 0 1 1 1 0 1 0 0 s
222	1	41	1	13	1 0 0 1 1 0 1 0 0 0 1 0 s
223	1	42	1	14	1 0 0 1 1 0 1 1 1 1 0 0 0 s
224	1	43	1	14	0 0 1 0 0 0 1 1 1 0 1 1 1 s
225	1	44	1	15	0 0 0 1 0 1 1 0 0 1 0 1 0 1 s
226	1	45	1	14	1 0 1 1 0 1 0 0 0 0 0 1 0 s
227	1	46	1	13	1 0 1 1 1 0 0 1 0 0 0 0 s
228	1	47	1	13	1 0 1 1 1 1 1 0 0 1 0 1 s
229	1	48	1	13	0 0 1 0 0 0 1 1 0 1 0 0 s
230	1	48	2	17	1 0 1 1 0 1 0 0 1 0 0 1 1 1 1 1 s
231	1	49	1	14	1 0 0 1 1 0 1 0 1 1 1 0 1 s
232	1	50	1	15	0 0 1 0 1 1 0 1 0 0 0 0 0 1 s
233	1	51	1	14	1 0 1 1 1 1 1 0 0 1 1 0 1 s
234	1	52	1	15	0 0 0 1 0 1 1 0 0 1 0 1 0 0 s
235	1	53	1	13	1 0 0 1 1 0 1 1 1 0 1 0 s
236	1	54	1	14	1 0 1 1 0 1 0 0 0 1 0 0 1 s
237	1	54	2	17	1 0 1 1 0 1 0 0 1 0 0 1 1 1 1 1 0 s
238	1	55	1	12	1 0 1 1 1 1 1 1 1 0 1 1 s
239	1	55	2	17	1 0 1 1 0 1 0 0 1 0 0 1 1 0 0 1 s
240	1	56	1	12	1 0 1 1 1 1 1 1 1 1 0 1 s
241	1	56	2	16	0 0 1 0 1 1 0 1 0 0 1 0 1 1 0 s
242	1	57	1	13	1 0 1 1 1 0 0 1 0 0 1 1 s
243	1	58	1	14	0 0 0 1 0 1 1 0 0 1 1 1 1 s
244	1	59	1	15	0 0 0 1 0 1 1 0 0 1 0 1 1 1 s
245	1	60	1	15	0 0 0 1 0 1 1 0 0 1 0 1 1 0 s
246	1	61	1	14	1 0 1 1 0 1 0 0 0 1 0 0 0 s
247	1	62	1	14	1 0 0 1 1 0 1 0 1 1 1 1 1 s
248	1	63	1	14	0 0 1 0 0 0 1 1 1 0 1 1 0 s

APPENDIX B. STATIC SLIDE VARIABLE LENGTH CODING TABLE (VLC)

The following is the custom VLC table used with static slide sequences. The last character in the codeword *s* indicates the appended sign bit.

INDEX	LAST	RUN	LEVEL	BITS	CODEWORD
1	0	0	1	4	101s
2	0	0	2	4	110s
3	0	0	3	5	1000s
4	0	0	4	7	111100s
5	0	0	5	7	001110s
6	0	0	6	6	00110s
7	0	0	7	13	0110110100011s
8	0	0	8	10	0111010011s
9	0	0	10	11	11111010011s
10	0	0	11	11	10010001100s
11	0	0	13	11	10011010000s
12	0	0	20	15	0011111011111100s
13	0	0	24	12	001111101100s
14	0	0	28	11	01101101001s
15	0	0	32	9	000011100s
16	0	0	36	9	011111100s
17	0	0	40	8	00001100s
18	0	0	44	8	10011100s
19	0	0	48	6	00011s
20	0	0	52	8	01100011s
21	0	0	56	8	00101011s
22	0	0	60	9	001000010s
23	0	0	64	4	010s
24	0	1	1	6	11100s
25	0	1	2	6	00010s
26	0	1	3	8	01110011s
27	0	1	4	10	1111111111s
28	0	1	5	10	0011111110s
29	0	1	6	9	001000011s
30	0	1	7	16	0011111011110010s
31	0	1	8	12	100100101111s
32	0	1	10	14	01111010010000s
33	0	1	13	15	111111110000101s
34	0	2	1	7	011100s
35	0	2	2	7	011010s
36	0	2	3	8	00100011s
37	0	2	4	11	01111011100s
38	0	2	5	9	100100011s
39	0	2	6	9	100111110s
40	0	2	7	17	00111101111111110s
41	0	2	8	12	001010011110s
42	0	2	10	14	11111111000001s
43	0	2	11	15	111111110011000s
44	0	2	13	14	11111111000011s
45	0	3	1	8	0111100 s
46	0	3	2	8	01100000s
47	0	3	3	9	000011101s

48	0	3	4	11	00001110 00s
49	0	3	5	10	00100101 0s
50	0	3	6	10	00111111 01s
51	0	3	7	17	00111101 11111001s
52	0	3	8	13	10010001 0111s
53	0	3	10	15	11111111 00111 00s
54	0	3	11	15	11111111 00110 11s
55	0	3	13	16	01101101 00101 11s
56	0	4	1	8	0110001s
57	0	4	2	8	0111111s
58	0	4	3	9	00100100s
59	0	4	4	10	10011010 1s
60	0	4	5	11	00001110 01s
61	0	4	6	9	00100011s
62	0	4	8	15	11111111 00100 00s
63	0	5	1	10	01111101 0s
64	0	5	2	9	01101100s
65	0	5	3	10	01111010 1s
66	0	5	4	12	10011111 1001s
67	0	5	5	12	00111100 010s
68	0	5	6	11	11111010 00s
69	0	5	8	15	11111111 00100 11s
70	0	6	1	9	11111011s
71	0	6	2	9	00001010s
72	0	6	3	10	10011010 11s
73	0	6	4	13	11111111 01111s
74	0	6	5	11	10010010 00s
75	0	6	6	12	01111011 011s
76	0	6	10	15	11111111 00111 11s
77	0	7	1	10	11110100 0 s
78	0	7	2	10	00100000 0s
79	0	7	3	11	10011010 01s
80	0	7	4	15	01101101 0010 10s
81	0	7	5	16	00100101 1000 011s
82	0	7	6	13	01110101 00000s
83	0	7	7	16	00111101 11111 101s
84	0	7	10	15	11111111 00111 10s
85	0	8	1	9	00101000s
86	0	8	2	9	00101111s
87	0	8	3	10	10011010 1 0s
88	0	8	4	11	00100101 11s
89	0	8	5	12	10011111 000s
90	0	8	6	10	01111011 1s
91	0	8	8	15	11111111 00100 10s
92	0	8	10	15	11111111 00110 01s
93	0	9	1	9	11110101s
94	0	9	2	9	10010000s
95	0	9	3	11	00101001 10s
96	0	9	4	15	00111101 11000 01s
97	0	9	5	11	11110100 10s
98	0	9	6	13	11111111 01110s
99	0	10	1	12	00111101 1101s
100	0	10	2	11	00100000 11s
101	0	10	3	13	00100000 01011s
102	0	10	4	15	00111101 11001 1s
103	0	10	5	13	00111101 00 11s

104	0	10	6	13	1 1 1 1 1 0 1 0 1 0 0 1 s
105	0	10	8	15	1 1 1 1 1 1 1 0 0 1 1 1 0 1 s
106	0	11	1	14	1 0 0 1 1 1 1 1 1 1 1 1 1 1 s
107	0	11	2	11	0 1 1 0 1 1 0 1 1 1 s
108	0	11	3	12	1 1 1 1 0 1 0 0 1 1 1 s
109	0	11	4	15	1 1 1 1 1 1 1 1 0 0 1 0 1 1 0 s
110	0	11	5	15	0 0 1 1 1 1 0 1 1 1 1 1 0 1 s
111	0	11	6	15	1 1 1 1 1 1 1 1 0 0 1 0 0 0 1 s
112	0	12	1	12	0 0 1 1 1 1 0 0 0 1 1 s
113	0	12	2	12	0 1 1 1 1 0 1 0 0 0 0 s
114	0	12	3	14	0 0 1 1 1 1 0 1 0 0 0 1 1 s
115	0	12	4	13	1 0 0 1 0 0 0 1 0 1 0 1 s
116	0	12	5	16	0 0 1 1 1 1 0 1 1 1 1 0 0 0 1 s
117	0	12	6	13	0 1 1 1 0 1 0 1 0 0 1 0 s
118	0	13	1	13	1 0 0 1 1 1 1 1 1 1 1 0 s
119	0	13	2	12	0 0 1 0 1 0 0 1 1 1 1 s
120	0	13	3	13	0 0 1 0 0 0 0 0 1 0 1 0 s
121	0	13	4	15	0 0 1 1 1 1 0 1 1 1 0 0 1 0 s
122	0	13	5	16	0 0 1 1 1 1 0 1 1 1 1 0 0 1 1 s
123	0	14	3	15	1 1 1 1 1 1 1 1 0 0 1 0 1 0 0 s
124	0	14	5	14	0 1 1 1 0 1 0 1 0 0 0 1 1 s
125	1	0	1	6	1 1 1 0 1 s
126	1	0	2	6	0 0 0 0 0 s
127	1	0	3	7	1 0 0 1 0 1 s
128	1	0	4	9	0 0 0 0 1 0 1 1 s
129	1	0	5	9	0 0 0 0 1 1 1 1 s
130	1	0	6	8	0 1 1 0 1 1 1 s
131	1	0	7	14	1 0 0 1 0 0 0 1 0 1 1 0 0 s
132	1	0	8	12	0 0 1 0 0 0 0 0 1 0 0 s
133	1	0	10	12	1 1 1 1 1 0 1 0 0 1 0 s
134	1	0	11	13	1 1 1 1 1 0 1 0 1 0 1 1 s
135	1	0	13	14	1 1 1 1 1 1 1 0 0 0 0 0 0 s
136	1	0	32	13	0 0 1 1 1 1 0 1 0 0 1 0 s
137	1	0	36	13	0 0 1 1 1 1 0 1 1 1 0 1 s
138	1	0	40	12	1 0 0 1 0 0 1 0 1 1 0 s
139	1	0	44	14	0 0 1 1 1 1 0 1 1 1 1 0 1 s
140	1	0	48	10	0 0 1 1 1 1 1 0 0 s
141	1	0	52	12	0 0 1 0 0 1 0 1 1 0 1 s
142	1	0	56	12	0 1 1 1 1 0 1 0 0 1 1 s
143	1	0	60	13	1 1 1 1 1 0 1 0 1 0 1 0 s
144	1	0	64	8	0 1 1 0 0 1 0 s
145	1	1	1	8	1 1 1 1 1 0 0 s
146	1	1	2	8	1 1 1 1 1 1 0 s
147	1	1	3	10	0 0 0 0 1 1 1 0 1 s
148	1	1	4	14	0 0 1 1 1 1 0 1 0 0 0 1 0 s
149	1	1	5	13	0 0 1 1 1 1 0 1 0 0 0 0 s
150	1	1	6	13	0 0 1 0 0 1 0 1 1 0 0 1 s
151	1	1	7	17	0 0 1 1 1 1 0 1 1 1 1 1 1 1 1 1 s
152	1	2	1	9	0 1 1 1 0 1 0 0 s
153	1	2	2	9	1 0 0 1 1 1 0 1 s
154	1	2	3	11	0 0 1 1 1 1 0 0 1 0 s
155	1	2	4	15	0 1 1 1 1 0 1 1 0 1 0 0 1 0 s
156	1	2	5	13	0 1 1 1 0 1 0 1 0 0 1 1 s
157	1	2	6	14	0 1 1 1 0 1 0 1 0 0 0 1 0 s
158	1	2	8	14	0 0 1 0 0 1 0 1 1 0 0 0 1 s
159	1	2	13	15	1 1 1 1 1 1 1 0 0 0 0 1 0 0 s

160	1	3	1	8	0 0 1 0 1 1 0 s
161	1	3	2	8	1 1 1 1 0 1 1 s
162	1	3	3	9	1 0 0 1 1 1 0 0 s
163	1	3	4	12	0 1 1 1 1 1 1 0 1 1 0 0 s
164	1	3	5	11	1 0 0 1 1 1 1 1 0 1 s
165	1	3	6	11	0 0 1 1 1 1 1 1 1 1 0 s
166	1	3	7	17	0 0 1 1 1 1 0 1 1 1 1 1 1 1 1 0 0 0 s
167	1	3	11	15	1 1 1 1 1 1 1 0 0 1 1 0 1 0 s
168	1	4	1	11	0 0 1 1 1 1 0 0 0 0 s
169	1	4	2	10	0 0 1 0 1 0 0 1 0 s
170	1	4	3	13	0 1 1 1 1 0 1 1 0 1 0 1 s
171	1	4	4	14	1 0 0 1 0 0 0 1 0 1 1 0 1 s
172	1	4	5	17	0 0 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 s
173	1	4	6	14	0 1 1 1 1 0 1 0 0 1 0 0 1 s
174	1	5	1	10	0 0 1 0 1 1 1 0 0 s
175	1	5	2	10	1 1 1 1 1 1 1 1 0 s
176	1	5	3	14	1 0 0 1 1 1 1 1 1 1 1 1 1 0 s
177	1	6	1	11	1 0 0 1 0 0 1 0 1 0 s
178	1	6	2	12	1 1 1 1 1 1 1 1 0 1 1 0 s
179	1	6	3	15	0 1 1 1 1 0 1 1 0 1 0 0 1 1 s
180	1	6	5	14	1 1 1 1 1 1 1 1 0 0 0 1 1 1 s
181	1	7	1	10	0 0 1 0 1 1 1 0 1 s
182	1	7	2	10	1 0 0 1 0 0 0 1 1 s
183	1	7	3	14	0 1 1 1 1 0 1 1 0 1 0 0 0 s
184	1	8	1	11	1 0 0 1 1 1 1 1 1 1 0 s
185	1	8	2	12	0 1 1 1 0 1 0 1 0 1 0 s
186	1	8	4	16	0 1 1 0 1 1 0 1 0 0 0 1 0 1 1 0 s
187	1	8	6	14	1 1 1 1 1 1 1 1 0 0 0 1 1 0 s
188	1	9	1	11	0 1 1 1 1 1 1 0 1 1 1 s
189	1	9	2	12	0 1 1 1 1 0 1 0 0 0 1 s
190	1	9	3	14	0 1 1 0 1 1 0 1 0 0 1 0 0 s
191	1	9	4	15	0 0 1 1 1 1 1 0 1 1 1 0 0 0 0 s
192	1	9	5	16	0 0 1 0 0 1 0 1 1 0 0 0 0 1 0 s
193	1	10	1	11	0 0 1 1 1 1 1 0 0 1 1 s
194	1	10	2	15	0 0 1 0 0 1 0 1 1 0 0 0 0 0 s
195	1	11	1	11	1 1 1 1 1 1 1 1 0 1 0 s
196	1	11	2	11	0 0 1 1 1 1 1 0 1 0 1 s
197	1	11	3	12	1 0 0 1 1 1 1 1 1 1 1 0 s
198	1	11	4	14	1 1 1 1 1 1 1 1 0 0 0 1 0 0 s
199	1	11	5	17	0 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 0 s
200	1	11	6	13	1 1 1 1 1 1 0 1 0 1 0 0 0 s
201	1	12	1	11	0 0 1 1 1 1 1 1 1 1 1 s
202	1	12	2	11	0 1 1 0 1 1 1 0 1 1 0 s
203	1	12	3	12	1 1 1 1 0 1 0 0 1 1 0 s
204	1	12	4	13	1 0 0 1 0 0 0 1 0 1 0 0 s
205	1	12	5	16	0 0 1 1 1 1 1 0 1 1 1 1 1 0 0 0 0 s
206	1	12	6	13	0 1 1 1 1 1 0 1 0 0 1 0 1 s
207	1	13	1	12	0 1 1 0 1 1 0 1 0 0 0 s
208	1	13	2	12	1 1 1 1 1 1 0 1 0 0 1 1 s
209	1	13	3	15	1 1 1 1 1 1 1 1 0 0 1 0 1 0 1 s
210	1	14	1	12	0 1 1 1 0 1 0 1 0 1 1 s
211	1	14	2	11	1 0 0 1 0 0 1 0 0 1 s
212	1	14	3	15	1 1 1 1 1 1 1 1 0 0 1 0 1 1 1 s
213	1	15	2	12	0 1 1 1 1 1 1 0 1 1 0 1 s
214	1	15	3	14	1 1 1 1 1 1 1 1 0 0 0 1 0 1 s

APPENDIX C. MATLAB CODE LIBRARY

This appendix contains the Matlab code used in the layered video codec. The main code block, *thesis.m*, is provided first, and the supporting functions follow in alphabetical order. As provided, *thesis.m* processes four video sequences: two motion video sequences of 100 frames each, followed by two, static slide sequences of 50 frames each.

```
% thesis

format compact
clear all
close all


global QD_TABLE VLC_DYN VLC_STA RV HUFF_TABLE LAST

load('g:\QD_TABLE')
load('g:\VlC_dyn')
load('g:\VLC_sta')

VLC_DYN = VLC_dyn; clear VLC_dyn                % gets custom VLC table for motion video
VLC_STA = VLC_sta; clear VLC_sta                % gets custom VLC table for static
slides

rand('state',0)                                 % seed for reproductivity
RV = floor(21*rand(99,1));                      % for aging algorithm
HUFF_TABLE = make_HAC_table;                    % gets JPEG standard table
LAST = ones(99,1);                              % sets block selection to 1 initially
threshold = 160;                                % for use with asd
count = -1;                                     % where is it in the loop?
mse = 1;                                        % calculate MSE?
display = 1;                                    % show the images?
write = 1;                                      % write to file for evaluation?
last = 299;                                     % last frame # to consider
tOff = 0;                                       % intialize (used with scene change)
frame_type = 0;                                 % initialize (zero means dynamic scene)
m_mat_ndx = [];                                 % initialize (track macroblocks
selected)
f_vec = zeros(1584,16);                         % initialize (reshaped frame)

f_far_ll = double(zeros(792,8));                % initialize for decoder
f_far_lh = double(zeros(792,8));                % initialize for decoder
f_far_hl = double(zeros(792,8));                % initialize for decoder
f_far_hh = double(zeros(792,8));                % initialize for decoder

f_far_ll_p1 = double(zeros(792,8));             % initialize for decoder
f_far_lh_p1 = double(zeros(792,8));             % initialize for decoder
f_far_hl_p1 = double(zeros(792,8));             % initialize for decoder
f_far_hh_p1 = double(zeros(792,8));             % initialize for decoder

f_far_ll_p2 = double(zeros(792,8));             % initialize for decoder
f_far_lh_p2 = double(zeros(792,8));             % initialize for decoder
f_far_hl_p2 = double(zeros(792,8));             % initialize for decoder
f_far_hh_p2 = double(zeros(792,8));             % initialize for decoder

f_1 = double(zeros(1584,16));                   % initialize for decoder
f_2 = double(zeros(1584,16));                   % initialize for decoder
f_3 = double(zeros(1584,16));                   % initialize for decoder
```

```
beta = -11.346;                                % slope of rate control curve

br = input('Enter the target bitrate (Kbps) \n >> ')
B_bar = br * 1024/10;
qd_entry = min(find(B_bar > QD_TABLE(:,4)));   % intial triplet based on test averages


for i = 0:last                                 % loop through frames

   count = count+1

   if (i<=99)
      i_present = get_next_image(i);           % read head#.bmp
   elseif ( (i>=100) & (i<=199) )
      i_present = get_next_image_2(i-100);     % read ncaa#.bmp
   elseif (i==200)
      i_present = get_next_image(101);         % read busy text slide
   elseif (i==250)
      i_present = get_next_image(102);         % read line drawing slide
   else
      i_present;
   end

   if display                                  % diplays original if desired
      figure;
      subplot(2,2,1)
      image(i_present)
      colormap(gray(256))
      title(['Original'])
      axis off
   end

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Coder %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   %%%%%%%%%% Change dimensions of image and identify macroblocks by threshold %%%%%%%%

   f_last = f_vec;                             % buffers previous frame
   f_vec = shape(i_present);                   % shapes current frame
   [m_vec_ndx,frame_type] = m_blk_id_xr(f_last,f_vec,threshold); % compare current to
last
   m_mat_ndx = [m_mat_ndx,m_vec_ndx];          % matrix of MB's selected

   if (sum(m_vec_ndx) >= 65)                   % triggers a scene change
      tOff = 1;                                % "trigger" flag set "on"
   end

   if (tOff)                                   % scene change just occured
      tOff = 0;                                % resets "trigger" flag
      flag = 1;                                % "flag" for 1st frame after scene change
      q1 = 64;                                 % heavily compressed scene change
      q2 = 1000;
      q3 = 1000;
   elseif frame_type
      q1 = 4;                                  % triplet for static sequence
      q2 = 16;
      q3 = 16;
   else                                        % a dynamic frame sequence
      delta_Binter = B_bar - (sum(m_vec_ndx)/sum(m_mat_ndx(:,i))) * total(i);
      qd_entry = get_qd_entryf(flag,B_bar,qd_entry,delta_Binter,sum(m_vec_ndx),beta);
      flag = 0;                                % resets flag
      q1 = QD_TABLE(qd_entry,1);               % rate control triplets fetched
      q2 = QD_TABLE(qd_entry,2);
      q3 = QD_TABLE(qd_entry,3);
   end

   Q = get_Q1_matrix(q1);                      % quantizer matrix (via JPEG standard)

   if ~frame_type                              % if dynamic frame sequence
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Transforms %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[f_fht_ll,f_fht_lh,f_fht_hl,f_fht_hh ] = fht(f_vec,m_vec_ndx,16);  % FHT of frame

f_fht_ll = f_fht_ll - 128;                                  % level shift of LL
f_fht_ll = dct_of_fht(f_fht_ll,m_vec_ndx);                  % 2_D DCT of LL

[f_fht_lh_ll,f_fht_lh_lh,f_fht_lh_hl,f_fht_lh_hh] = ...
   fht(f_fht_lh,m_vec_ndx,8);                               % subband LH FHT

[f_fht_hl_ll,f_fht_hl_lh,f_fht_hl_hl,f_fht_hl_hh] = ...
   fht(f_fht_hl,m_vec_ndx,8);                               % subband HL FHT

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Quantizing %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

f_fht_ll_q = quantizer_ll(f_fht_ll,Q,m_vec_ndx);            % quantize LL

f_fht_hh_q = round(f_fht_hh/q3);                            % quantize HH

f_fht_lh_ll_q = round(f_fht_lh_ll/q2);                      % quantize of LH subbands
f_fht_lh_lh_q = round(f_fht_lh_lh/q2);
f_fht_lh_hl_q = round(f_fht_lh_hl/q3);
f_fht_lh_hh_q = round(f_fht_lh_hh/q3);

f_fht_hl_ll_q = round(f_fht_hl_ll/q2);                      % quantize of HL subbands
f_fht_hl_lh_q = round(f_fht_hl_lh/q3);
f_fht_hl_hl_q = round(f_fht_hl_hl/q2);
f_fht_hl_hh_q = round(f_fht_hl_hh/q3);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%% Working on LL %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% zig-zag scans each LL 8x8 (Results are one 6336x1 vector [ll_zz] and the index
%              of the last non-zero entity [last_ll_zz].  Get "inf" if a group of
%              64 is all zeros.)
[last_ll_zz,ll_zz] = zzb(f_fht_ll_q,8);

% gets rid of trailing zeros (one big vector of varying size.)
a_ll_zz = make_it_compact(ll_zz,last_ll_zz,8);

% parsing LL with Huffman routine
parsed_ll_zz = parse_Huff(a_ll_zz,last_ll_zz);

% gets bits per frame due to LL
bits_ll_zz(i+1) = get_bits_Huff(parsed_ll_zz);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%% Working on HH %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% scans each HH 8x8 (Results are one 6336x1 vector [hh_r] and the index of the last
%              non-zero entity [last_hh_r], where r indicates horizontal raster
%              the scan method.  Get "inf" if a group of 64 is all zeros.)
[last_hh_r,hh_r] = raster(f_fht_hh_q,8);

% gets rid of trailing zeros (one big vector of varying size)
a_hh_r = make_it_compact(hh_r,last_hh_r,8);

% parsing HH and getting the number of parsings for later used in eval_thesis to
%              get the percent of time that the default bit number is used.
parsed_hh_r_3D = parse_3D(a_hh_r,last_hh_r);
l_p_hh_r_3D(i+1) = length(parsed_hh_r_3D(:,1));

% gets bits per frame and number of times the default is chosen
%              (includes 99 COD bits)
[bits_hh_r_3D(i+1), hh_r_22(i+1)] = get_bits2(parsed_hh_r_3D,frame_type);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%% Working on LH subbands %%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%% Doing the LH_LL subband %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% scans each LH_LL 4x4 (Results are one 1584x1 vector [lh_ll_r] and the index of
%              the last non-zero entity [last_lh_ll_r], where the r indicates
%              horizontal raster scan method.  Get "inf" if a group of 16 is all
%              zeros.)
[last_lh_ll_r,lh_ll_r] = raster(f_fht_lh_ll_q,4);

% gets rid of trailing zeros (one big vector of varying size)
a_lh_ll_r = make_it_compact(lh_ll_r,last_lh_ll_r,4);

% parsing LH_LL and getting the number of parsings for later used in eval_thesis to
%              get the percent of time that the default bit number is used.
parsed_lh_ll_r_3D = parse_3D(a_lh_ll_r,last_lh_ll_r);
l_p_lh_ll_r_3D(i+1) = length(parsed_lh_ll_r_3D(:,1));

% gets bits per frame and number of times the default is chosen
%              (includes 99 COD bits)
[bits_lh_ll_r_3D(i+1), lh_ll_r_22(i+1)] = get_bits2(parsed_lh_ll_r_3D,frame_type);

%%%%%%%%%%%%%%%%%%%%%%%%%% Doing the LH_LH subband %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% scans each LH_LH 4x4 (Results are one 1584x1 vector [lh_lh_r] and the index of
%              the last non-zero entity [last_lh_lh_r], where the r indicates
%              horizontal raster scan method.  Get "inf" if a group of 16 is all
%              zeros.)
[last_lh_lh_r,lh_lh_r] = raster(f_fht_lh_lh_q,4);

% gets rid of trailing zeros (one big vector of varying size)
a_lh_lh_r = make_it_compact(lh_lh_r,last_lh_lh_r,4);

% parsing LH_LH and getting the number of parsings for later used in eval_thesis to
%              get the percent of time that the default bit number is used.
parsed_lh_lh_r_3D = parse_3D(a_lh_lh_r,last_lh_lh_r);
l_p_lh_lh_r_3D(i+1) = length(parsed_lh_lh_r_3D(:,1));

% gets bits per frame and number of times the default is chosen
%              (includes 99 COD bits)
[bits_lh_lh_r_3D(i+1), lh_lh_r_22(i+1)] = get_bits2(parsed_lh_lh_r_3D,frame_type);

%%%%%%%%%%%%%%%%%%%%%%%%%% Doing the LH_HL subband %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% scans each LH_HL 4x4 (Results are 1584x1 vector [lh_hl_r] and the index of
%              the last non-zero entity [last_lh_hl_r], where the r indicates
%              horizontal raster scan method.  Get "inf" if a group of 16 is all
%              zeros.)
[last_lh_hl_r,lh_hl_r] = raster(f_fht_lh_hl_q,4);

% gets rid of trailing zeros (one big vector of varying size)
a_lh_hl_r = make_it_compact(lh_hl_r,last_lh_hl_r,4);

% parsing LH_HL and getting the number of parsings for later used in eval_thesis to
%              get the percent of time that the default bit number is used.
parsed_lh_hl_r_3D = parse_3D(a_lh_hl_r,last_lh_hl_r);
l_p_lh_hl_r_3D(i+1) = length(parsed_lh_hl_r_3D(:,1));

% gets bits per frame and number of times the default is chosen
%              (includes 99 COD bits)
[bits_lh_hl_r_3D(i+1), lh_hl_r_22(i+1)] = get_bits2(parsed_lh_hl_r_3D,frame_type);

%%%%%%%%%%%%%%%%%%%%%%%%%% Doing the LH_HH subband %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% scans each LH_HH 4x4 (Results are 1584x1 vector [lh_hh_r] and the index of
%              the last non-zero entity [last_lh_hh_r], where the r indicates
%              horizontal raster scan method.  Get "inf" if a group of 16 is all
%              zeros.)
[last_lh_hh_r,lh_hh_r] = raster(f_fht_lh_hh_q,4);

% gets rid of trailing zeros (one big vector of varying size)
```

```
a_lh_hh_r = make_it_compact(lh_hh_r,last_lh_hh_r,4);

% parsing LH_HH and getting the number of parsings for later used in eval_thesis to
%              get the percent of time that the default bit number is used.
parsed_lh_hh_r_3D = parse_3D(a_lh_hh_r,last_lh_hh_r);
l_p_lh_hh_r_3D(i+1) = length(parsed_lh_hh_r_3D(:,1));

% gets bits per frame and number of times the default is chosen
%              (includes 99 COD bits)
[bits_lh_hh_r_3D(i+1), lh_hh_r_22(i+1)] = get_bits2(parsed_lh_hh_r_3D,frame_type);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%% Working on HL subbands %%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%% Doing the HL_LL subband %%%%%%%%%%%%%%%%%%%%%%%%%%%%

% scans each HL_LL 4x4 (Results are a 1584x1 vector [hl_ll_v] and the index of
%              the last non-zero entity [last_hl_ll_v], where the v indicates
%              vertical raster scan method.  Get "inf" if a group of 16 is all
%              zeros.)
[last_hl_ll_v,hl_ll_v] = vertical(f_fht_hl_ll_q,4);

% gets rid of trailing zeros (one big vector of varying size)
a_hl_ll_v = make_it_compact(hl_ll_v,last_hl_ll_v,4);

% parsing HL_LL and getting the number of parsings for later used in eval_thesis to
%              get the percent of time that the default bit number is used.
parsed_hl_ll_v_3D = parse_3D(a_hl_ll_v,last_hl_ll_v);
l_p_hl_ll_v_3D(i+1) = length(parsed_hl_ll_v_3D(:,1));

% gets bits per frame and number of times the default is chosen
%              (includes 99 COD bits)
[bits_hl_ll_v_3D(i+1), hl_ll_v_22(i+1)] = get_bits2(parsed_hl_ll_v_3D,frame_type);

%%%%%%%%%%%%%%%%%%%%%%%%%%%% Doing the HL_LH subband %%%%%%%%%%%%%%%%%%%%%%%%%%%%

% scans each HL_LH 4x4 (Results are a 1584x1 vector [hl_lh_v] and the index of
%              the last non-zero entity [last_hl_lh_v], where the v indicates
%              vertical raster scan method.  Get "inf" if a group of 16 is all
%              zeros.)
[last_hl_lh_v,hl_lh_v] = vertical(f_fht_hl_lh_q,4);

% gets rid of trailing zeros (one big vector of varying size)
a_hl_lh_v = make_it_compact(hl_lh_v,last_hl_lh_v,4);

% parsing HL_LH and getting the number of parsings for later used in eval_thesis to
%              get the percent of time that the default bit number is used.
parsed_hl_lh_v_3D = parse_3D(a_hl_lh_v,last_hl_lh_v);
l_p_hl_lh_v_3D(i+1) = length(parsed_hl_lh_v_3D(:,1));

% gets bits per frame and number of times the default is chosen
%              (includes 99 COD bits)
[bits_hl_lh_v_3D(i+1), hl_lh_v_22(i+1)] = get_bits2(parsed_hl_lh_v_3D,frame_type);

%%%%%%%%%%%%%%%%%%%%%%%%%%%% Doing the HL_HL subband %%%%%%%%%%%%%%%%%%%%%%%%%%%%

% scans each HL_HL 4x4 (Results are a 1584x1 vector [hl_hl_v] and the index of
%              the last non-zero entity [last_hl_hl_v], where the v indicates
%              vertical raster scan method.  Get "inf" if a group of 16 is all
%              zeros.)
[last_hl_hl_v,hl_hl_v] = vertical(f_fht_hl_hl_q,4);

% gets rid of trailing zeros (one big vector of varying size)
a_hl_hl_v = make_it_compact(hl_hl_v,last_hl_hl_v,4);

% parsing HL_HL and getting the number of parsings for later used in eval_thesis to
%              get the percent of time that the default bit number is used.
parsed_hl_hl_v_3D = parse_3D(a_hl_hl_v,last_hl_hl_v);
l_p_hl_hl_v_3D(i+1) = length(parsed_hl_hl_v_3D(:,1));
```

```
% gets bits per frame and number of times the default is chosen
%              (includes 99 COD bits)
[bits_hl_hl_v_3D(i+1), hl_hl_v_22(i+1)] = get_bits2(parsed_hl_hl_v_3D,frame_type);

%%%%%%%%%%%%%%%%%%%%%%%%%%% Doing the HL_HH subband %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% scans each HL_HL 4x4 (Results are a 1584x1 vector [hl_hh_v] and the index of
%              the last non-zero entity [last_hl_hh_v], where the v indicates
%              vertical raster scan method.  Get "inf" if a group of 16 is all
%              zeros.)
[last_hl_hh_v,hl_hh_v] = vertical(f_fht_hl_hh_q,4);

% gets rid of trailing zeros (one big vector of varying size)
a_hl_hh_v = make_it_compact(hl_hh_v,last_hl_hh_v,4);

% parsing HL_HH and getting the number of parsings for later used in eval_thesis to
%              get the percent of time that the default bit number is used.
parsed_hl_hh_v_3D = parse_3D(a_hl_hh_v,last_hl_hh_v);
l_p_hl_hh_v_3D(i+1) = length(parsed_hl_hh_v_3D(:,1));

% gets bits per frame and number of times the default is chosen
%              (includes 99 COD bits)
[bits_hl_hh_v_3D(i+1), hl_hh_v_22(i+1)] = get_bits2(parsed_hl_hh_v_3D,frame_type);

% getting bits per frame per layer
layer1(i+1) = bits_ll_zz(i+1);
layer2(i+1) = bits_lh_ll_r_3D(i+1) + bits_lh_lh_r_3D(i+1) + ...
   bits_hl_ll_v_3D(i+1) + bits_hl_hl_v_3D(i+1);
layer3(i+1) = bits_hh_r_3D(i+1) + bits_lh_hl_r_3D(i+1) + bits_lh_hh_r_3D(i+1) + ...
   bits_hl_lh_v_3D(i+1) + l_p_hl_hh_v_3D(i+1);
total(i+1) = layer1(i+1) + layer2(i+1) + layer3(i+1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                                  %
%                                  Channel                                         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Unquantize %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

f_fht_ll_uq = unquantize_ll(f_fht_ll_q,Q,m_vec_ndx); % unquantize LL

f_fht_hh_uq = f_fht_hh_q * q3;                       % unquantize HH

f_fht_lh_ll_uq = f_fht_lh_ll_q * q2;                 % unquantize of LH subbands
f_fht_lh_lh_uq = f_fht_lh_lh_q * q2;
f_fht_lh_hl_uq = f_fht_lh_hl_q * q3;
f_fht_lh_hh_uq = f_fht_lh_hh_q * q3;

f_fht_hl_ll_uq = f_fht_hl_ll_q * q2;                 % unquantize of HL subbands
f_fht_hl_lh_uq = f_fht_hl_lh_q * q3;
f_fht_hl_hl_uq = f_fht_hl_hl_q * q2;
f_fht_hl_hh_uq = f_fht_hl_hh_q * q3;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Inverse Transform %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

f_far_ll = invdct_of_fht(f_fht_ll_uq,m_vec_ndx);    % inv 2-D dct of LL
f_far_ll = f_far_ll + 128;                          % level shift
f_far_lh = remake_3(f_far_lh,f_fht_lh_ll_uq,f_fht_lh_lh_uq,f_fht_lh_hl_uq, ...
   f_fht_lh_hh_uq,m_vec_ndx,8);                     % LH subband inv FHT
f_far_hl = remake_3(f_far_hl,f_fht_hl_ll_uq,f_fht_hl_lh_uq,f_fht_hl_hl_uq, ...
   f_fht_hl_hh_uq,m_vec_ndx,8);                     % HL subband inv FHT
f_far_hh = f_fht_hh_uq;

f_far_ll_p2 = f_far_ll;                             % LL p2 assignment
f_far_lh_p2 = remake_3(f_far_lh_p2,f_fht_lh_ll_uq,f_fht_lh_lh_uq,...
   0,0,m_vec_ndx,8);                                % LH p2 subband inv FHT
f_far_hl_p2 = remake_3(f_far_hl_p2,f_fht_hl_ll_uq,0,...
   f_fht_hl_hl_uq,0,m_vec_ndx,8);                   % HL p2 subband inv FHT
```

```
   f_far_hh_p2 = 0;                                    % HH p2 assignment

   f_far_ll_p1 = f_far_ll;                             % LL p1 assignment
   f_far_lh_p1 = 0;                                    % HH p1 assignment
   f_far_hl_p1 = 0;                                    % HH p1 assignment
   f_far_hh_p1 = 0;                                    % HH p1 assignment


   f_3 = remake_3(f_3,f_far_ll,f_far_lh,f_far_hl,...
      f_far_hh,m_vec_ndx,16);                          % frame inv FHT with 3 layers
   f_2 = remake_3(f_2,f_far_ll_p2,f_far_lh_p2,f_far_hl_p2,...
      f_far_hh_p2,m_vec_ndx,16);                       % frame inv FHT with 2 layers
   f_1 = remake_3(f_1,f_far_ll_p1,f_far_lh_p1,f_far_hl_p1,...
      f_far_hh_p1,m_vec_ndx,16);                       % frame inv FHT with 1 layer

else                                                   % a staic frame sequence

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Transforms %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   [f_fht_ll,f_fht_lh,f_fht_hl,f_fht_hh ] = fht(f_vec,m_vec_ndx,16);  % FHT of frame

   [f_fht_ll_ll,f_fht_ll_lh,f_fht_ll_hl,f_fht_ll_hh] = ...
      fht(f_fht_ll,m_vec_ndx,8);                       % subband LL FHT

   [f_fht_lh_ll,f_fht_lh_lh,f_fht_lh_hl,f_fht_lh_hh] = ...
      fht(f_fht_lh,m_vec_ndx,8);                       % subband LH FHT

   [f_fht_hl_ll,f_fht_hl_lh,f_fht_hl_hl,f_fht_hl_hh] = ...
      fht(f_fht_hl,m_vec_ndx,8);                       % subband HL FHT

   [f_fht_hh_ll,f_fht_hh_lh,f_fht_hh_hl,f_fht_hh_hh] = ...
      fht(f_fht_hh,m_vec_ndx,8);                       % subband HH FHT

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Quantizing %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   f_fht_ll_ll_q = round(f_fht_ll_ll/q1);              % quantize of LL subbands
   f_fht_ll_lh_q = round(f_fht_ll_lh/q2);
   f_fht_ll_hl_q = round(f_fht_ll_hl/q2);
   f_fht_ll_hh_q = round(f_fht_ll_hh/q3);

   f_fht_lh_ll_q = round(f_fht_lh_ll/q2);              % quantize of LH subbands
   f_fht_lh_lh_q = round(f_fht_lh_lh/q2);
   f_fht_lh_hl_q = round(f_fht_lh_hl/q3);
   f_fht_lh_hh_q = round(f_fht_lh_hh/q3);

   f_fht_hl_ll_q = round(f_fht_hl_ll/q2);              % quantize of HL subbands
   f_fht_hl_lh_q = round(f_fht_hl_lh/q3);
   f_fht_hl_hl_q = round(f_fht_hl_hl/q2);
   f_fht_hl_hh_q = round(f_fht_hl_hh/q3);

   f_fht_hh_ll_q = round(f_fht_hh_ll/q3);              % quantize of HL subbands
   f_fht_hh_lh_q = round(f_fht_hh_lh/q3);
   f_fht_hh_hl_q = round(f_fht_hh_hl/q3);
   f_fht_hh_hh_q = round(f_fht_hh_hh/q3);


   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %%%%%%%%%%%%%%%%%%%%%%%%%%%% Working on LL %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   %%%%%%%%%%%%%%%%%%%%%%%%%%%% Doing the LL_LL subband %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   % scans each LL_LL 4x4 (Results are one 1584x1 vector [ll_ll_r] and the index of
   %              the last non-zero entity [last_ll_ll_r], where the r indicates
   %              horizontal raster scan method.  Get "inf" if a group of 16 is all
   %              zeros.)
   [last_ll_ll_r,ll_ll_r] = raster(f_fht_ll_ll_q,4);

   % gets rid of trailing zeros (one big vector of varying size)
   a_ll_ll_r = make_it_compact(ll_ll_r,last_ll_ll_r,4);
```

```
% parsing LL_LL and getting the number of parsings for later used in eval_thesis to
%              get the percent of time that the default bit number is used.
parsed_ll_ll_r_3D = parse_3D(a_ll_ll_r,last_ll_ll_r);
l_p_ll_ll_r_3D(i+1) = length(parsed_ll_ll_r_3D(:,1));

% gets bits per frame and number of times the default is chosen
%              (includes 99 COD bits)
[bits_ll_ll_r_3D(i+1), ll_ll_r_22(i+1)] = get_bits2(parsed_ll_ll_r_3D,frame_type);

%%%%%%%%%%%%%%%%%%%%%%%%%%% Doing the LL_LH subband %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% scans each LL_LH 4x4 (Results are one 1584x1 vector [ll_lh_r] and the index of
%              the last non-zero entity [last_ll_lh_r], where the r indicates
%              horizontal raster scan method.  Get "inf" if a group of 16 is all
%              zeros.)
[last_ll_lh_r,ll_lh_r] = raster(f_fht_ll_lh_q,4);

% gets rid of trailing zeros (one big vector of varying size)
a_ll_lh_r = make_it_compact(ll_lh_r,last_ll_lh_r,4);

% parsing LL_LH and getting the number of parsings for later used in eval_thesis to
%              get the percent of time that the default bit number is used.
parsed_ll_lh_r_3D = parse_3D(a_ll_lh_r,last_ll_lh_r);
l_p_ll_lh_r_3D(i+1) = length(parsed_ll_lh_r_3D(:,1));

% gets bits per frame and number of times the default is chosen
%              (includes 99 COD bits)
[bits_ll_lh_r_3D(i+1), ll_lh_r_22(i+1)] = get_bits2(parsed_ll_lh_r_3D,frame_type);

%%%%%%%%%%%%%%%%%%%%%%%%%%% Doing the LL_HL subband %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% scans each LL_HL 4x4 (Results are 1584x1 vector [ll_hl_r] and the index of
%              the last non-zero entity [last_ll_hl_r], where the r indicates
%              horizontal raster scan method.  Get "inf" if a group of 16 is all
%              zeros.)
[last_ll_hl_r,ll_hl_r] = raster(f_fht_ll_hl_q,4);

% gets rid of trailing zeros (one big vector of varying size)
a_ll_hl_r = make_it_compact(ll_hl_r,last_ll_hl_r,4);

% parsing LL_HL and getting the number of parsings for later used in eval_thesis to
%              get the percent of time that the default bit number is used.
parsed_ll_hl_r_3D = parse_3D(a_ll_hl_r,last_ll_hl_r);
l_p_ll_hl_r_3D(i+1) = length(parsed_ll_hl_r_3D(:,1));

% gets bits per frame and number of times the default is chosen
%              (includes 99 COD bits)
[bits_ll_hl_r_3D(i+1), ll_hl_r_22(i+1)] = get_bits2(parsed_ll_hl_r_3D,frame_type);

%%%%%%%%%%%%%%%%%%%%%%%%%%% Doing the LL_HH subband %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% scans each LL_HH 4x4 (Results are 1584x1 vector [ll_hh_r] and the index of
%              the last non-zero entity [last_ll_hh_r], where the r indicates
%              horizontal raster scan method.  Get "inf" if a group of 16 is all
%              zeros.)

[last_ll_hh_r,ll_hh_r] = raster(f_fht_ll_hh_q,4);

% gets rid of trailing zeros (one big vector of varying size)
a_ll_hh_r = make_it_compact(ll_hh_r,last_ll_hh_r,4);

% parsing LL_HH and getting the number of parsings for later used in eval_thesis to
%              get the percent of time that the default bit number is used.
parsed_ll_hh_r_3D = parse_3D(a_ll_hh_r,last_ll_hh_r);
l_p_ll_hh_r_3D(i+1) = length(parsed_ll_hh_r_3D(:,1));

% gets bits per frame and number of times the default is chosen
%              (includes 99 COD bits)

[bits_ll_hh_r_3D(i+1), ll_hh_r_22(i+1)] = get_bits2(parsed_ll_hh_r_3D,frame_type);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%% Working on LH subbands %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%% Doing the LH_LL subband %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% scans each LH_LL 4x4 (Results are one 1584x1 vector [lh_ll_r] and the index of
%             the last non-zero entity [last_lh_ll_r], where the r indicates
%             horizontal raster scan method.  Get "inf" if a group of 16 is all
%             zeros.)
[last_lh_ll_r,lh_ll_r] = raster(f_fht_lh_ll_q,4);

% gets rid of trailing zeros (one big vector of varying size)
a_lh_ll_r = make_it_compact(lh_ll_r,last_lh_ll_r,4);

% parsing LH_LL and getting the number of parsings for later used in eval_thesis to
%             get the percent of time that the default bit number is used.

parsed_lh_ll_r_3D = parse_3D(a_lh_ll_r,last_lh_ll_r);
l_p_lh_ll_r_3D(i+1) = length(parsed_lh_ll_r_3D(:,1));

% gets bits per frame and number of times the default is chosen
%             (includes 99 COD bits)
[bits_lh_ll_r_3D(i+1), lh_ll_r_22(i+1)] = get_bits2(parsed_lh_ll_r_3D,frame_type);

%%%%%%%%%%%%%%%%%%%%%%%%%%% Doing the LH_LH subband %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% scans each LH_LH 4x4 (Results are one 1584x1 vector [lh_lh_r] and the index of
%             the last non-zero entity [last_lh_lh_r], where the r indicates
%             horizontal raster scan method.  Get "inf" if a group of 16 is all
%             zeros.)
[last_lh_lh_r,lh_lh_r] = raster(f_fht_lh_lh_q,4);

% gets rid of trailing zeros (one big vector of varying size)
a_lh_lh_r = make_it_compact(lh_lh_r,last_lh_lh_r,4);

% parsing LH_LH and getting the number of parsings for later used in eval_thesis to
%             get the percent of time that the default bit number is used.
parsed_lh_lh_r_3D = parse_3D(a_lh_lh_r,last_lh_lh_r);
l_p_lh_lh_r_3D(i+1) = length(parsed_lh_lh_r_3D(:,1));

% gets bits per frame and number of times the default is chosen
%             (includes 99 COD bits)
[bits_lh_lh_r_3D(i+1), lh_lh_r_22(i+1)] = get_bits2(parsed_lh_lh_r_3D,frame_type);

%%%%%%%%%%%%%%%%%%%%%%%%%%% Doing the LH_HL subband %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% scans each LH_HL 4x4 (Results are 1584x1 vector [lh_hl_r] and the index of
%             the last non-zero entity [last_lh_hl_r], where the r indicates
%             horizontal raster scan method.  Get "inf" if a group of 16 is all
%             zeros.)
[last_lh_hl_r,lh_hl_r] = raster(f_fht_lh_hl_q,4);

% gets rid of trailing zeros (one big vector of varying size)
a_lh_hl_r = make_it_compact(lh_hl_r,last_lh_hl_r,4);

% parsing LH_HL and getting the number of parsings for later used in eval_thesis to
%             get the percent of time that the default bit number is used.
parsed_lh_hl_r_3D = parse_3D(a_lh_hl_r,last_lh_hl_r);
l_p_lh_hl_r_3D(i+1) = length(parsed_lh_hl_r_3D(:,1));

% gets bits per frame and number of times the default is chosen
%             (includes 99 COD bits)
[bits_lh_hl_r_3D(i+1), lh_hl_r_22(i+1)] = get_bits2(parsed_lh_hl_r_3D,frame_type);

%%%%%%%%%%%%%%%%%%%%%%%%%%% Doing the LH_HH subband %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% scans each LH_HH 4x4 (Results are 1584x1 vector [lh_hh_r] and the index of
%             the last non-zero entity [last_lh_hh_r], where the r indicates
%             horizontal raster scan method.  Get "inf" if a group of 16 is all
```

```
%              zeros.)
[last_lh_hh_r,lh_hh_r] = raster(f_fht_lh_hh_q,4);

% gets rid of trailing zeros (one big vector of varying size)
a_lh_hh_r = make_it_compact(lh_hh_r,last_lh_hh_r,4);

% parsing LH_HH and getting the number of parsings for later used in eval_thesis to
%              get the percent of time that the default bit number is used.
parsed_lh_hh_r_3D = parse_3D(a_lh_hh_r,last_lh_hh_r);
l_p_lh_hh_r_3D(i+1) = length(parsed_lh_hh_r_3D(:,1));

% gets bits per frame and number of times the default is chosen
%              (includes 99 COD bits)
[bits_lh_hh_r_3D(i+1), lh_hh_r_22(i+1)] = get_bits2(parsed_lh_hh_r_3D,frame_type);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%% Working on HL subbands %%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%% Doing the HL_LL subband %%%%%%%%%%%%%%%%%%%%%%%%%%%%

% scans each HL_LL 4x4 (Results are a 1584x1 vector [hl_ll_v] and the index of
%              the last non-zero entity [last_hl_ll_v], where the v indicates
%              vertical raster scan method.  Get "inf" if a group of 16 is all
%              zeros.)
[last_hl_ll_v,hl_ll_v] = vertical(f_fht_hl_ll_q,4);

% gets rid of trailing zeros (one big vector of varying size)
a_hl_ll_v = make_it_compact(hl_ll_v,last_hl_ll_v,4);

% parsing HL_LL and getting the number of parsings for later used in eval_thesis to
%              get the percent of time that the default bit number is used.
parsed_hl_ll_v_3D = parse_3D(a_hl_ll_v,last_hl_ll_v);
l_p_hl_ll_v_3D(i+1) = length(parsed_hl_ll_v_3D(:,1));

% gets bits per frame and number of times the default is chosen
%              (includes 99 COD bits)
[bits_hl_ll_v_3D(i+1), hl_ll_v_22(i+1)] = get_bits2(parsed_hl_ll_v_3D,frame_type);

%%%%%%%%%%%%%%%%%%%%%%%%%%%% Doing the HL_LH subband %%%%%%%%%%%%%%%%%%%%%%%%%%%%

% scans each HL_LH 4x4 (Results are a 1584x1 vector [hl_lh_v] and the index of
%              the last non-zero entity [last_hl_lh_v], where the v indicates
%              vertical raster scan method.  Get "inf" if a group of 16 is all
%              zeros.)
[last_hl_lh_v,hl_lh_v] = vertical(f_fht_hl_lh_q,4);

% gets rid of trailing zeros (one big vector of varying size)
a_hl_lh_v = make_it_compact(hl_lh_v,last_hl_lh_v,4);

% parsing HL_LH and getting the number of parsings for later used in eval_thesis to
%              get the percent of time that the default bit number is used.
parsed_hl_lh_v_3D = parse_3D(a_hl_lh_v,last_hl_lh_v);
l_p_hl_lh_v_3D(i+1) = length(parsed_hl_lh_v_3D(:,1));

% gets bits per frame and number of times the default is chosen
%              (includes 99 COD bits)
[bits_hl_lh_v_3D(i+1), hl_lh_v_22(i+1)] = get_bits2(parsed_hl_lh_v_3D,frame_type);

%%%%%%%%%%%%%%%%%%%%%%%%%%%% Doing the HL_HL subband %%%%%%%%%%%%%%%%%%%%%%%%%%%%

% scans each HL_HL 4x4 (Results are a 1584x1 vector [hl_hl_v] and the index of
%              the last non-zero entity [last_hl_hl_v], where the v indicates
%              vertical raster scan method.  Get "inf" if a group of 16 is all
%              zeros.)
[last_hl_hl_v,hl_hl_v] = vertical(f_fht_hl_hl_q,4);

% gets rid of trailing zeros (one big vector of varying size)
a_hl_hl_v = make_it_compact(hl_hl_v,last_hl_hl_v,4);
```

```
% parsing HL_HL and getting the number of parsings for later used in eval_thesis to
%              get the percent of time that the default bit number is used.
parsed_hl_hl_v_3D = parse_3D(a_hl_hl_v,last_hl_hl_v);
l_p_hl_hl_v_3D(i+1) = length(parsed_hl_hl_v_3D(:,1));

% gets bits per frame and number of times the default is chosen
%              (includes 99 COD bits)
[bits_hl_hl_v_3D(i+1), hl_hl_v_22(i+1)] = get_bits2(parsed_hl_hl_v_3D,frame_type);

%%%%%%%%%%%%%%%%%%%%%%%%%%%% Doing the HL_HH subband %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% scans each HL_HL 4x4 (Results are a 1584x1 vector [hl_hh_v] and the index of
%              the last non-zero entity [last_hl_hh_v], where the v indicates
%              vertical raster scan method.  Get "inf" if a group of 16 is all
%              zeros.)
[last_hl_hh_v,hl_hh_v] = vertical(f_fht_hl_hh_q,4);

% gets rid of trailing zeros (one big vector of varying size)
a_hl_hh_v = make_it_compact(hl_hh_v,last_hl_hh_v,4);

% parsing HL_HH and getting the number of parsings for later used in eval_thesis to
%              get the percent of time that the default bit number is used.
parsed_hl_hh_v_3D = parse_3D(a_hl_hh_v,last_hl_hh_v);
l_p_hl_hh_v_3D(i+1) = length(parsed_hl_hh_v_3D(:,1));

% gets bits per frame and number of times the default is chosen
%              (includes 99 COD bits)
[bits_hl_hh_v_3D(i+1), hl_hh_v_22(i+1)] = get_bits2(parsed_hl_hh_v_3D,frame_type);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%% Working on HH subbands %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%% Doing the HH_LL subband %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% scans each HH_LL 4x4 (Results are one 1584x1 vector [hh_ll_r] and the index of
%              the last non-zero entity [last_hh_ll_r], where the r indicates
%              horizontal raster scan method.  Get "inf" if a group of 16 is all
%              zeros.)
[last_hh_ll_r,hh_ll_r] = raster(f_fht_hh_ll_q,4);

% gets rid of trailing zeros (one big vector of varying size)
a_hh_ll_r = make_it_compact(hh_ll_r,last_hh_ll_r,4);

% parsing HH_LL and getting the number of parsings for later used in eval_thesis to
%              get the percent of time that the default bit number is used.
parsed_hh_ll_r_3D = parse_3D(a_hh_ll_r,last_hh_ll_r);
l_p_hh_ll_r_3D(i+1) = length(parsed_hh_ll_r_3D(:,1));

% gets bits per frame and number of times the default is chosen
%              (includes 99 COD bits)
[bits_hh_ll_r_3D(i+1), hh_ll_r_22(i+1)] = get_bits2(parsed_hh_ll_r_3D,frame_type);

%%%%%%%%%%%%%%%%%%%%%%%%%%%% Doing the HH_LH subband %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% scans each HH_LH 4x4 (Results are one 1584x1 vector [hh_lh_r] and the index of
%              the last non-zero entity [last_hh_lh_r], where the r indicates
%              horizontal raster scan method.  Get "inf" if a group of 16 is all
%              zeros.)
[last_hh_lh_r,hh_lh_r] = raster(f_fht_hh_lh_q,4);

% gets rid of trailing zeros (one big vector of varying size)
a_hh_lh_r = make_it_compact(hh_lh_r,last_hh_lh_r,4);

% parsing HH_LH and getting the number of parsings for later used in eval_thesis to
%              get the percent of time that the default bit number is used.
parsed_hh_lh_r_3D = parse_3D(a_hh_lh_r,last_hh_lh_r);
l_p_hh_lh_r_3D(i+1) = length(parsed_hh_lh_r_3D(:,1));

% gets bits per frame and number of times the default is chosen
%              (includes 99 COD bits)
```

```
[bits_hh_lh_r_3D(i+1), hh_lh_r_22(i+1)] = get_bits2(parsed_hh_lh_r_3D,frame_type);

%%%%%%%%%%%%%%%%%%%%%%%%%%%% Doing the HH_HL subband %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% scans each HH_HL 4x4 (Results are 1584x1 vector [hh_hl_r] and the index of
%               the last non-zero entity [last_hh_hl_r], where the r indicates
%               horizontal raster scan method.  Get "inf" if a group of 16 is all
%               zeros.)
[last_hh_hl_r,hh_hl_r] = raster(f_fht_hh_hl_q,4);

% gets rid of trailing zeros (one big vector of varying size)
a_hh_hl_r = make_it_compact(hh_hl_r,last_hh_hl_r,4);

% parsing HH_HL and getting the number of parsings for later used in eval_thesis to
%               get the percent of time that the default bit number is used.
parsed_hh_hl_r_3D = parse_3D(a_hh_hl_r,last_hh_hl_r);
l_p_hh_hl_r_3D(i+1) = length(parsed_hh_hl_r_3D(:,1));

% gets bits per frame and number of times the default is chosen
%               (includes 99 COD bits)
[bits_hh_hl_r_3D(i+1), hh_hl_r_22(i+1)] = get_bits2(parsed_hh_hl_r_3D,frame_type);

%%%%%%%%%%%%%%%%%%%%%%%%%%%% Doing the HH_HH subband %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% scans each HH_HH 4x4 (Results are 1584x1 vector [hh_hh_r] and the index of
%               the last non-zero entity [last_hh_hh_r], where the r indicates
%               horizontal raster scan method.  Get "inf" if a group of 16 is all
%               zeros.)
[last_hh_hh_r,hh_hh_r] = raster(f_fht_hh_hh_q,4);

% gets rid of trailing zeros (one big vector of varying size)
a_hh_hh_r = make_it_compact(hh_hh_r,last_hh_hh_r,4);

% parsing HH_HH and getting the number of parsings for later used in eval_thesis to
%               get the percent of time that the default bit number is used.

parsed_hh_hh_r_3D = parse_3D(a_hh_hh_r,last_hh_hh_r);
l_p_hh_hh_r_3D(i+1) = length(parsed_hh_hh_r_3D(:,1));

% gets bits per frame and number of times the default is chosen
%               (includes 99 COD bits)
[bits_hh_hh_r_3D(i+1), hh_hh_r_22(i+1)] = get_bits2(parsed_hh_hh_r_3D,frame_type);

% getting bits per frame per layer and total
layer1(i+1) = bits_ll_ll_r_3D(i+1) + bits_ll_lh_r_3D(i+1) + ...
   bits_lh_ll_r_3D(i+1)+ bits_lh_lh_r_3D(i+1) + ...
   bits_ll_hl_r_3D(i+1)+ bits_hl_ll_v_3D(i+1) + ...
   bits_hh_ll_r_3D(i+1)+ bits_hl_hl_v_3D(i+1) + ...
   bits_hh_hh_r_3D(i+1);
layer2(i+1) = bits_ll_hh_r_3D(i+1) + bits_lh_hl_r_3D(i+1) + ...
   bits_lh_hh_r_3D(i+1) +  bits_hl_lh_v_3D(i+1) + ...
   bits_hl_hh_v_3D(i+1);
layer3(i+1) = bits_hh_lh_r_3D(i+1) + bits_hh_hl_r_3D(i+1);
total(i+1) = layer1(i+1) + layer2(i+1) + layer3(i+1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                                        %
%                                     Channel                                            %
%                                                                                        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Unquantize %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

f_fht_ll_ll_uq = f_fht_ll_ll_q * q1;                     % unquantize of LH subbands
f_fht_ll_lh_uq = f_fht_ll_lh_q * q2;
f_fht_ll_hl_uq = f_fht_ll_hl_q * q2;
f_fht_ll_hh_uq = f_fht_ll_hh_q * q3;

f_fht_lh_ll_uq = f_fht_lh_ll_q * q2;                     % unquantize of LH subbands
f_fht_lh_lh_uq = f_fht_lh_lh_q * q2;
f_fht_lh_hl_uq = f_fht_lh_hl_q * q3;
```

```
   f_fht_lh_hh_uq = f_fht_lh_hh_q * q3;

   f_fht_hl_ll_uq = f_fht_hl_ll_q * q2;                    % unquantize of HL subbands
   f_fht_hl_lh_uq = f_fht_hl_lh_q * q3;
   f_fht_hl_hl_uq = f_fht_hl_hl_q * q2;
   f_fht_hl_hh_uq = f_fht_hl_hh_q * q3;

   f_fht_hh_ll_uq = f_fht_hh_ll_q * q3;                    % unquantize of HH subbands
   f_fht_hh_lh_uq = f_fht_hh_lh_q * q3;
   f_fht_hh_hl_uq = f_fht_hh_hl_q * q3;
   f_fht_hh_hh_uq = f_fht_hh_hh_q * q3;

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Inverse Transform %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   f_far_ll = remake_3(f_far_ll,f_fht_ll_ll_uq,f_fht_ll_lh_uq,f_fht_ll_hl_uq, ...
      f_fht_ll_hh_uq,m_vec_ndx,8);                         % LL subband inv FHT
   f_far_lh = remake_3(f_far_lh,f_fht_lh_ll_uq,f_fht_lh_lh_uq,f_fht_lh_hl_uq, ...
      f_fht_lh_hh_uq,m_vec_ndx,8);                         % LH subband inv FHT
   f_far_hl = remake_3(f_far_hl,f_fht_hl_ll_uq,f_fht_hl_lh_uq,f_fht_hl_hl_uq, ...
      f_fht_hl_hh_uq,m_vec_ndx,8);                         % HL subband inv FHT
   f_far_hh = remake_3(f_far_hh,f_fht_hh_ll_uq,f_fht_hh_lh_uq,f_fht_hh_hl_uq, ...
      f_fht_hh_hh_uq,m_vec_ndx,8);                         % HH subband inv FHT


   f_far_ll_p2 = remake_3(f_far_ll_p2,f_fht_ll_ll_uq,f_fht_ll_lh_uq,...
      f_fht_ll_hl_uq,f_fht_ll_hh_uq,m_vec_ndx,8);          % LL p2 subband inv FHT
   f_far_lh_p2 = remake_3(f_far_lh_p2,f_fht_lh_ll_uq,f_fht_lh_lh_uq,...
      f_fht_lh_hl_uq,f_fht_lh_hh_uq,m_vec_ndx,8);          % LH p2 subband inv FHT
   f_far_hl_p2 = remake_3(f_far_hl_p2,f_fht_hl_ll_uq,f_fht_hl_lh_uq,...
      f_fht_hl_hl_uq,f_fht_hl_hh_uq,m_vec_ndx,8);          % HL p2 subband inv FHT
   f_far_hh_p2 = remake_3(f_far_hh_p2,f_fht_hh_ll_uq,0,0,...
      f_fht_hh_hh_uq,m_vec_ndx,8);                         % HH p2 subband inv FHT


   f_far_ll_p1 = remake_3(f_far_ll_p1,f_fht_ll_ll_uq,f_fht_ll_lh_uq,...
      f_fht_ll_hl_uq,0,m_vec_ndx,8);                       % LL p1 subband inv FHT
   f_far_lh_p1 = remake_3(f_far_lh_p1,f_fht_lh_ll_uq,f_fht_lh_lh_uq,...
      0,0,m_vec_ndx,8);                                    % LH p1 subband inv FHT
   f_far_hl_p1 = remake_3(f_far_hl_p1,f_fht_hl_ll_uq,0,...
      f_fht_hl_hl_uq,0,m_vec_ndx,8);                       % HL p1 subband inv FHT
   f_far_hh_p1 = remake_3(f_far_hh_p1,f_fht_hh_ll_uq,0,...
      0,f_fht_hh_hh_uq,m_vec_ndx,8);                       % HH p1 subband inv FHT


   f_3 = remake_3(f_3,f_far_ll,f_far_lh,f_far_h1,...
      f_far_hh,m_vec_ndx,16);                              % frame inv FHT with 3 layers
   f_2 = remake_3(f_2,f_far_ll_p2,f_far_lh_p2,f_far_hl_p2,...
      f_far_hh_p2,m_vec_ndx,16);                           % frame inv FHT with 2 layers
   f_1 = remake_3(f_1,f_far_ll_p1,f_far_lh_p1,f_far_hl_p1,...
      f_far_hh_p1,m_vec_ndx,16);                           % frame inv FHT with 1 layer

end % ends frame_type descrimination

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Make Display Size %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fig_3 = shape_back(f_3);                                   % make display dimensions
fig_2 = shape_back(f_2);                                   % make display dimensions
fig_1 = shape_back(f_1);                                   % make display dimensions

L3 = round(fig_3);                                         % for viewing
L2 = round(fig_2);                                         % for viewing
L1 = round(fig_1);                                         % for viewing

%%%%%%%%%%%%%%%%%%%%%%%% Calculates the mean-square-error per frame %%%%%%%%%%%%%%%%%
if mse
   MSE_3L(i+1,1) = (sum(sum((i_present - fig_3).^2)))/176/144;
   MSE_2L(i+1,1) = (sum(sum((i_present - fig_2).^2)))/176/144;
   MSE_1L(i+1,1) = (sum(sum((i_present - fig_1).^2)))/176/144;
end % end if mse
```

```
   if display

      % The following lines show the figure with 3 layers
      subplot(2,2,2)
      image(L3)
      axis off
      title('Layers 1, 2, and 3')

      % The following lines show the figure with 2 layers
      subplot(2,2,3)
      image(L2)
      title('Layers 1 and 2')
      axis off

      % The following lines show the figure with 1 layer
      subplot(2,2,4)
      image(L1)
      title('Layer 1 Only')
      axis off
      drawnow

   end % ends if display

   % closing reconstruction occasionally to conserve memory
   if ((i==20)|(i==40)|(i==60)|(i==80)|(i==100)|(i==120)|(i==140)|(i==160)|(i==180)|...
      (i==200)|(i==220)|(i==240)|(i==260)|(i==280))
      close all
   end
end % ends looping through frames

% The following lines simply rename variables to be consistent with an off-line
% evaluation program used in the development.

% LL dynamic
BITS_LL_HUFF = bits_ll_zz';

% LL static
BITS_LL_LL_3D = bits_ll_ll_r_3D';
SCAN_LL_LL_22 = ll_ll_r_22';
SCAN_LL_LL_LEN = l_p_ll_ll_r_3D';

BITS_LL_LH_3D = bits_ll_lh_r_3D';
SCAN_LL_LH_22 = ll_lh_r_22';
SCAN_LL_LH_LEN = l_p_ll_lh_r_3D';

BITS_LL_HL_3D = bits_ll_hl_r_3D';
SCAN_LL_HL_22 = ll_hl_r_22';
SCAN_LL_HL_LEN = l_p_ll_hl_r_3D';

BITS_LL_HH_3D = bits_ll_hh_r_3D';
SCAN_LL_HH_22 = ll_hh_r_22';
SCAN_LL_HH_LEN = l_p_ll_hh_r_3D';

% LH both types of slide
BITS_LH_LL_3D = bits_lh_ll_r_3D';
SCAN_LH_LL_22 = lh_ll_r_22';
SCAN_LH_LL_LEN = l_p_lh_ll_r_3D';

BITS_LH_LH_3D = bits_lh_lh_r_3D';
SCAN_LH_LH_22 = lh_lh_r_22';
SCAN_LH_LH_LEN = l_p_lh_lh_r_3D';

BITS_LH_HL_3D = bits_lh_hl_r_3D';
SCAN_LH_HL_22 = lh_hl_r_22';
SCAN_LH_HL_LEN = l_p_lh_hl_r_3D';

BITS_LH_HH_3D = bits_lh_hh_r_3D';
SCAN_LH_HH_22 = lh_hh_r_22';
SCAN_LH_HH_LEN = l_p_lh_hh_r_3D';

% HL both types of slides
```

```
BITS_HL_LL_3D = bits_hl_ll_v_3D';
SCAN_HL_LL_22 = hl_ll_v_22';
SCAN_HL_LL_LEN = l_p_hl_ll_v_3D';

BITS_HL_LH_3D = bits_hl_lh_v_3D';
SCAN_HL_LH_22 = hl_lh_v_22';
SCAN_HL_LH_LEN = l_p_hl_lh_v_3D';

BITS_HL_HL_3D = bits_hl_hl_v_3D';
SCAN_HL_HL_22 = hl_hl_v_22';
SCAN_HL_HL_LEN = l_p_hl_hl_v_3D';

BITS_HL_HH_3D = bits_hl_hh_v_3D';
SCAN_HL_HH_22 = hl_hh_v_22';
SCAN_HL_HH_LEN = l_p_hl_hh_v_3D';

% HH from dynamic
BITS_HH_3D = bits_hh_r_3D';
SCAN_HH_22 = hh_r_22';
SCAN_HH_LEN = l_p_hh_r_3D';

% HH from static
BITS_HH_LL_3D = bits_hh_ll_r_3D';
SCAN_HH_LL_22 = hh_ll_r_22';
SCAN_HH_LL_LEN = l_p_hh_ll_r_3D';

BITS_HH_LH_3D = bits_hh_lh_r_3D';
SCAN_HH_LH_22 = hh_lh_r_22';
SCAN_HH_LH_LEN = l_p_hh_lh_r_3D';

BITS_HH_HL_3D = bits_hh_hl_r_3D';
SCAN_HH_HL_22 = hh_hl_r_22';
SCAN_HH_HL_LEN = l_p_hh_hl_r_3D';

BITS_HH_HH_3D = bits_hh_hh_r_3D';
SCAN_HH_HH_22 = hh_hh_r_22';
SCAN_HH_HH_LEN = l_p_hh_hh_r_3D';

% saves parameters for evaluation later
if write
   s = char('g:\thesis\BR');
   s = strcat(s,num2str(br));
   save(s, 'm_mat_ndx','BITS_*','MSE_*','SCAN_*','layer*')
end % if write
```

```
function [output] = dct_of_fht(input,m_vec_ndx)

% performs the 2D DCT of input.  m_vec_ndx identifies where this operation needs
% to be performed.

output = zeros(792,8);
offset = -8;

for ndx = 1:99
   offset = offset + 8;

   if m_vec_ndx(ndx)
      output(offset+1:offset+8,:) = dct2(input(offset+1:offset+8,:));
   end

end
```

```
function [f_fht_ll,f_fht_lh,f_fht_hl,f_fht_hh] = fht(f_vec,m_vec_ndx,in)

% Performs the FHT of the appropriate IN x IN macroblocks of F_VEC as
% specified by M_VEC_NDX. Returns four sets of 99 IN/2 x IN/2 matrices,
% each stacked into one big 99*IN/2 x IN/2 matrix. The places
% where the fht was not performed are filled with zeroes as place-holders.

half = in/2;
f_fht_ll = zeros(99*half,half);
f_fht_lh = zeros(99*half,half);
f_fht_hl = zeros(99*half,half);
f_fht_hh = zeros(99*half,half);
mask_lh = [1 1;-1 -1];
mask_hl = [1 -1;1 -1];
mask_hh = [1 -1;-1 1];

offset = -in;

for ndx = 1:99
   offset= offset+in;
   oso2 = offset/2;

   if m_vec_ndx(ndx)

      for rndx = 1:half

         for cndx = 1:half
            f_fht_ll(rndx+oso2,cndx) = ...
               sum(sum(f_vec(rndx*2-1+offset:rndx*2+offset,cndx*2-1:cndx*2)))/4;
            f_fht_lh(rndx+oso2,cndx) = sum(sum(mask_lh .* ...
               f_vec(rndx*2-1+offset:rndx*2+offset,cndx*2-1:cndx*2)))/4;
            f_fht_hl(rndx+oso2,cndx) = sum(sum(mask_hl .* ...
               f_vec(rndx*2-1+offset:rndx*2+offset,cndx*2-1:cndx*2)))/4;
            f_fht_hh(rndx+oso2,cndx) = sum(sum(mask_hh .* ...
               f_vec(rndx*2-1+offset:rndx*2+offset,cndx*2-1:cndx*2)))/4;

         end

      end

   end

end
```

```
function [bits] = get_bits_Huff(parsed)

% Uses Huffman table to get bits for JPEG-based compression of PARSED.

global HUFF_TABLE

bits = 0;
[r,c] = size(parsed);

for ndx = 1:r

   while(parsed(ndx,1) >= 15)
      bits = bits + 11;
      parsed(ndx,1) = parsed(ndx,1) - 15;
   end

   table_row = find((HUFF_TABLE(:,2) == parsed(ndx,1)) & ...
      (HUFF_TABLE(:,3) == parsed(ndx,2)));
   bits = bits + HUFF_TABLE(table_row,4) + parsed(ndx,2);
end
```

```
bits = bits + 99*4;




function [bits,count] = get_bits2(parsed,kind)

% Uses PARSED to fetch bits from custom VLC tables
% KIND will be 1 for static scenes
% KIND will be 0 for dynamic scenec

global VLC_DYN VLC_STA

bits = 0;
count = 0;
parsed = abs(parsed);
[r,c] = size(parsed);

if kind
   for ndx = 1:r
      poss_rows = find(VLC_STA(:,2) == parsed(ndx,1));
      start = poss_rows(1);
      poss_rows = find(VLC_STA((poss_rows(1):poss_rows(length(poss_rows))),3) ==...
         parsed(ndx,2));

      if ~(isempty(poss_rows))
         poss_rows = poss_rows + start - 1;
         start = poss_rows(1);
      end

      if ~(isempty(poss_rows))
         poss_rows = find(VLC_STA((poss_rows(1):poss_rows(length(poss_rows))),4) ==...
            parsed(ndx,3));
         poss_rows = poss_rows + start - 1;
      end

      if ~(isempty(poss_rows))
         bits = bits + VLC_STA(poss_rows,5);
      else
         bits = bits + 22;
         count = count + 1;
      end

   end

else

   for ndx = 1:r
      poss_rows = find(VLC_DYN(:,2) == parsed(ndx,1));
      start = poss_rows(1);
      poss_rows = find(VLC_DYN((poss_rows(1):poss_rows(length(poss_rows))),3) ==...
      parsed(ndx,2));

      if ~(isempty(poss_rows))
         poss_rows = poss_rows + start - 1;
         start = poss_rows(1);
      end

      if ~(isempty(poss_rows))
         poss_rows = find(VLC_DYN((poss_rows(1):poss_rows(length(poss_rows))),4) ==...
         parsed(ndx,3));
         poss_rows = poss_rows + start - 1;
      end

      if ~(isempty(poss_rows))
         bits = bits + VLC_DYN(poss_rows,5);
      else
         bits = bits + 22;
         count = count + 1;
```

```
      end

   end

end
bits = bits + 99;
```

```
function [f] = get_next_image(num)

% Gets next frame as a .bmp file converts to form needed in MATLAB

s = char('g:\pictures\Head');
fn = strcat(s,num2str(num));
f = imread(fn,'bmp');
f = double(f(:,:,1));
```

```
function [f] = get_next_image_2(num)

% Gets next frame as a .bmp file converts to form needed in MATLAB

s = char('g:\pictures\ncaa');
fn = strcat(s,num2str(num));
f = imread(fn,'bmp');
f = double(f(:,:,1));
```

```
function [Q] = get_Q1_matrix(q1)

% makes the JPEG standard quantization matrix and multiplies it by q1.
% q1 = 16 will result in no scaling of the matrix when coupled with the rest
% of the code.  q1 < 16 is finer quantization, i.e. less quantization noise results.

Q = [16 11 10 16 24 40 51 61;
   12 12 14 19 26 58 60 55;
   14 13 16 24 40 57 69 56;
   14 17 22 29 51 87 80 62;
   18 22 37 56 68 109 103 77;
   24 35 55 64 81 104 113 92;
   49 64 78 87 103 121 120 101;
   72 92 95 98 112 100 103 99] .* q1;
```

```
function [out] = get_qd_entryf(flag,default,entry,delta_Binter,MBnum,beta)

% selects the appropriate quantizer triplet based on the input parameters
% FLAG implies the first frame following a scene change frame.  DEFAULT is the
% choice of triplet based on test sequences and serves as a starting point for a
% new sequence.  ENTRY hold the table entry from the previous frame.  The remaining
% parameters ara as defined in the thesis.

global QD_TABLE

if (flag)
```

```
   out = min(find(default > QD_TABLE(:,4)));
else
   deltaQ = fix(delta_Binter/MBnum/beta);
   out = entry + deltaQ;
   if (out > 17)
      out = 17;
   end
   if (out < 1)
      out = 1;
   end
 end
```

```
function [run] = get_run(seq_to_code,len)

% Called by the parsing functions PARSE_3D and PARSE_HUFF, this function obtains
% the RUN field for RLE.

run = zeros(len,1);

count = 0;
place = 1;
mdx = 1;

while (mdx <= length(seq_to_code))
   if seq_to_code(mdx)
      place = place + 1;
      mdx = mdx + 1;
   else
      while (seq_to_code(mdx) == 0)
         mdx = mdx + 1;
         count = count + 1;
      end
   run(place) = count;
   count = 0;
   place = place + 1;
   mdx = mdx + 1;
   end
end
```

```
function [output] = invdct_of_fht(input,m_vec_ndx)

% Performs the inverse 2D DCT.  M_VEC_NDX identifies where this operation needs to be
% performed.

output = zeros(792,8);
offset = -8;

for ndx = 1:99
   offset = offset + 8;

   if m_vec_ndx(ndx)
      output(offset+1:offset+8,:) = idct2(input(offset+1:offset+8,:));
   end

end
```

```
function [m_vec_ndx,byAge,b_count,mb_count] = m_blk_id_xr(f1,f2,T)

% Figures out which 16x16 macroblocks of the 144x176 image need to be further
% processed via an absolute sum of differences between the last frame f1 and
% the current frame f2. m_vec_ndx is 99x1 vector of 0 or 1 raster scanned.
% T is the threshold utilized.  byAge is 1 if macroblocks are selected only due
% to aging.  byAge is 0 otherwise.  b_count and mb_count were used in refining the
% search sequence.  b_count is "block count."  mb_count is "macroblock count."  As
% are artifacts now, they are commented-out.

global LAST RV

%b_count = 0;
%mb_count = 0;
r_count = 0;
m_vec_ndx = zeros(99,1);

for ndx = 1:99
   if (RV(ndx) == 0)
      m_vec_ndx(ndx) = 1;
      RV(ndx) = floor(21*rand);
   else
      RV(ndx) = RV(ndx) - 1;
   end
end

byAge1 = sum(m_vec_ndx);

for rndx = 1:16:1584
   r_count = r_count + 1;

   if (~m_vec_ndx(r_count))
      go = 1;
      if ((LAST(r_count) == 1) & (go))
         for cndx = 1
            f1_8x8 = f1(rndx:rndx+7,cndx:cndx+7);
            f2_8x8 = f2(rndx:rndx+7,cndx:cndx+7);
            asd = abs(sum(sum(f1_8x8-f2_8x8)));

            if (asd > T)
               m_vec_ndx(r_count) = 1; % ID's position
               %mb_count = mb_count + 1;
               %b_count = b_count + 1;
               LAST(r_count) = 1;
               go = 0;
               break; % gets out of the inner loop if justified now
            end

            f1_8x8 = f1(rndx+8:rndx+15,cndx+8:cndx+15);
            f2_8x8 = f2(rndx+8:rndx+15,cndx+8:cndx+15);
            asd = abs(sum(sum(f1_8x8-f2_8x8)));


            if (asd > T)
               m_vec_ndx(r_count) = 1; % ID's position
               %mb_count = mb_count + 1;
               %b_count = b_count + 2;
               LAST(r_count) = 3;
               go = 0;
               break; % gets out of the inner loop if justified now
            end

            f1_8x8 = f1(rndx:rndx+7,cndx+8:cndx+15);
            f2_8x8 = f2(rndx:rndx+7,cndx+8:cndx+15);
            asd = abs(sum(sum(f1_8x8-f2_8x8)));

            if (asd > T)
               m_vec_ndx(r_count) = 1; % ID's position
               %mb_count = mb_count + 1;
               %b_count = b_count + 3;
               LAST(r_count) = 2;
```

```
         go = 0;
         break; % gets out of the inner loop if justified now
      end

      f1_8x8 = f1(rndx+8:rndx+15,cndx:cndx+7);
      f2_8x8 = f2(rndx+8:rndx+15,cndx:cndx+7);
      asd = abs(sum(sum(f1_8x8-f2_8x8)));

      if (asd > T)
         m_vec_ndx(r_count) = 1; % ID's position
         %mb_count = mb_count + 1;
         %b_count = b_count + 4;
         LAST(r_count) = 4;
         go = 0;
      end

   end %for cndx
end % if LAST

if ((LAST(r_count) == 2) & (go))
   for cndx = 1
      f1_8x8 = f1(rndx:rndx+7,cndx+8:cndx+15);
      f2_8x8 = f2(rndx:rndx+7,cndx+8:cndx+15);
      asd = abs(sum(sum(f1_8x8-f2_8x8)));

      if (asd > T)
         m_vec_ndx(r_count) = 1; % ID's position
         %mb_count = mb_count + 1;
         %b_count = b_count + 1;
         LAST(r_count) = 2;
         go = 0;
         break; % gets out of the inner loop if justified now
      end

      f1_8x8 = f1(rndx+8:rndx+15,cndx:cndx+7);
      f2_8x8 = f2(rndx+8:rndx+15,cndx:cndx+7);
      asd = abs(sum(sum(f1_8x8-f2_8x8)));

      if (asd > T)
         m_vec_ndx(r_count) = 1; % ID's position
         %mb_count = mb_count + 1;
         %b_count = b_count + 2;
         LAST(r_count) = 4;
         go = 0;
         break; % gets out of the inner loop if justified now
      end

      f1_8x8 = f1(rndx+8:rndx+15,cndx+8:cndx+15);
      f2_8x8 = f2(rndx+8:rndx+15,cndx+8:cndx+15);
      asd = abs(sum(sum(f1_8x8-f2_8x8)));

      if (asd > T)
         m_vec_ndx(r_count) = 1; % ID's position
         %mb_count = mb_count + 1;
         %b_count = b_count + 3;
         LAST(r_count) = 3;
         go = 0;
         break; % gets out of the inner loop if justified now
      end

      f1_8x8 = f1(rndx:rndx+7,cndx:cndx+7);
      f2_8x8 = f2(rndx:rndx+7,cndx:cndx+7);
      asd = abs(sum(sum(f1_8x8-f2_8x8)));

      if (asd > T)
         m_vec_ndx(r_count) = 1; % ID's position
         %mb_count = mb_count + 1;
         %b_count = b_count + 4;
         LAST(r_count) = 1;
         go = 0;
      end
```

```
   end %for cndx
end % if LAST

if ((LAST(r_count) == 3) & (go))
   for cndx = 1
      f1_8x8 = f1(rndx+8:rndx+15,cndx+8:cndx+15);
      f2_8x8 = f2(rndx+8:rndx+15,cndx+8:cndx+15);
      asd = abs(sum(sum(f1_8x8-f2_8x8)));

      if (asd > T)
         m_vec_ndx(r_count) = 1; % ID's position
         %mb_count = mb_count + 1;
         %b_count = b_count + 1;
         LAST(r_count) = 3;
         go = 0;
         break; % gets out of the inner loop if justified now
      end

      f1_8x8 = f1(rndx:rndx+7,cndx:cndx+7);
      f2_8x8 = f2(rndx:rndx+7,cndx:cndx+7);
      asd = abs(sum(sum(f1_8x8-f2_8x8)));

      if (asd > T)
         m_vec_ndx(r_count) = 1; % ID's position
         %mb_count = mb_count + 1;
         %b_count = b_count + 2;
         LAST(r_count) = 1;
         go = 0;
         break; % gets out of the inner loop if justified now
      end

      f1_8x8 = f1(rndx+8:rndx+15,cndx:cndx+7);
      f2_8x8 = f2(rndx+8:rndx+15,cndx:cndx+7);
      asd = abs(sum(sum(f1_8x8-f2_8x8)));

      if (asd > T)
         m_vec_ndx(r_count) = 1; % ID's position
         %mb_count = mb_count + 1;
         %b_count = b_count + 3;
         LAST(r_count) = 4;
         go = 0;
         break; % gets out of the inner loop if justified now
      end

      f1_8x8 = f1(rndx:rndx+7,cndx+8:cndx+15);
      f2_8x8 = f2(rndx:rndx+7,cndx+8:cndx+15);
      asd = abs(sum(sum(f1_8x8-f2_8x8)));

      if (asd > T)
         m_vec_ndx(r_count) = 1; % ID's position
         %mb_count = mb_count + 1;
         %b_count = b_count + 4;
         LAST(r_count) = 2;
         go = 0;
      end

   end %for cndx
end % if LAST

if ((LAST(r_count) == 4) & (go))
   for cndx = 1
      f1_8x8 = f1(rndx+8:rndx+15,cndx:cndx+7);
      f2_8x8 = f2(rndx+8:rndx+15,cndx:cndx+7);
      asd = abs(sum(sum(f1_8x8-f2_8x8)));

      if (asd > T)
         m_vec_ndx(r_count) = 1; % ID's position
         %mb_count = mb_count + 1;
         %b_count = b_count + 1;
         LAST(r_count) = 4;
```

```
            go = 0;
            break; % gets out of the inner loop if justified now
         end

         f1_8x8 = f1(rndx:rndx+7,cndx+8:cndx+15);
         f2_8x8 = f2(rndx:rndx+7,cndx+8:cndx+15);
         asd = abs(sum(sum(f1_8x8-f2_8x8)));

         if (asd > T)
            m_vec_ndx(r_count) = 1; % ID's position
            %mb_count = mb_count + 1;
            %b_count = b_count + 2;
            LAST(r_count) = 2;
            go = 0;
            break; % gets out of the inner loop if justified now
         end

         f1_8x8 = f1(rndx:rndx+7,cndx:cndx+7);
         f2_8x8 = f2(rndx:rndx+7,cndx:cndx+7);
         asd = abs(sum(sum(f1_8x8-f2_8x8)));

         if (asd > T)
            m_vec_ndx(r_count) = 1; % ID's position
            %mb_count = mb_count + 1;
            %b_count = b_count + 3;
            LAST(r_count) = 1;
            go = 0;
            break; % gets out of the inner loop if justified now
         end

         f1_8x8 = f1(rndx+8:rndx+15,cndx+8:cndx+15);
         f2_8x8 = f2(rndx+8:rndx+15,cndx+8:cndx+15);
         asd = abs(sum(sum(f1_8x8-f2_8x8)));

         if (asd > T)
            m_vec_ndx(r_count) = 1; % ID's position
            %mb_count = mb_count + 1;
            %b_count = b_count + 4;
            LAST(r_count) = 3;
            go = 0;
         end

      end %for cndx
   end % if LAST

end %if ~m_vec_ndx

end % for rndx

byAge2 = sum(m_vec_ndx);
byAge = (byAge1 == byAge2);
```

```
function [table] = make_HAC_table();

% Generates the Huffman VLC table

vec1 = ones(10,1);
vec2 = (1:10)';

index = (1:162)';
runn = [0;vec1*0;vec1;vec1*2;vec1*3;vec1*4;vec1*5;vec1*6;
   vec1*7;vec1*8;vec1*9;vec1*10;vec1*11;vec1*12;vec1*13;
   vec1*14;vec1*15;15];
siz = [0;vec2;vec2;vec2;vec2;vec2;vec2;vec2;vec2;vec2;
   vec2;vec2;vec2;vec2;vec2;vec2;0;vec2];
cw_len = [0;2;2;3;4;5;7;8;10;16;16;4;5;7;9;11;16;16;16;16;
```

```
   16;5;8;10;12;ones(6,1)*16;6;9;12;ones(7,1)*16;6;10;
   ones(8,1)*16;8;11;ones(8,1)*16;7;12;ones(8,1)*16;8;
   12;ones(8,1)*16;9;ones(9,1)*16;9;ones(9,1)*16;9;
   ones(9,1)*16;10;ones(9,1)*16;10;ones(9,1)*16;11;
   ones(19,1)*16;11;ones(10,1)*16];
table = [index, runn, siz, cw_len];
```

```
function [a] = make_it_compact(big,index,size)

% gets rid of trailing zeros in the vector BIG that resulted from scanning 99 matrices
% of dimensions SIZExSIZE. INDEX holds the position of the last non-zero entry in each of
% the 99 matrices.


entry = size^2;
offset = -entry;
a = [];
for ndx = 1:99
   offset = entry + offset;
   if isinf(index(ndx))
      a = [a ; inf];
   else
      a = [a ; [big(offset+1:offset+index(ndx))]];
   end
end
```

```
function [parsed] = parse_3D(vector,index)

% RLE's VECTOR into the {last,run,level} format.  INDEX is the position of the last non
% zero entity in each of the 99 matrices.

last = [];
level = [];
run = [];

for ndx = 1:99
   if isinf(index(ndx))
      index(ndx) = 1;
   end
   point = sum(index(1:ndx));
   seq_to_code = vector(point-index(ndx)+1:point);
   len = length(seq_to_code);
   if ((len == 1) & isinf(seq_to_code))
      last = [last;0];
      level = [level;0];
      run = [run;0];
   else
      dummy = seq_to_code(find(seq_to_code));
      level = [level;dummy];
      last = [last;zeros(length(dummy)-1,1);1];
      this_run = get_run(seq_to_code,length(dummy));
      run = [run;this_run];
   end
end

parsed = [last,run,level];
```

```
function [parsed] = parse_Huff(vector,index)

% Parses VECTOR into the JPEG format. INDEX holds the position of the last non-zero
% value of each of the 99 matrices.

last = [];
level = [];
run = [];

for ndx = 1:99
   if isinf(index(ndx))
      index(ndx) = 1;
   end
   point = sum(index(1:ndx));
   seq_to_code = vector(point-index(ndx)+1:point);
   len = length(seq_to_code);
   if ((len == 1) & isinf(seq_to_code))
      level = [level;0];
      run = [run;0];
   else
      dummy = seq_to_code(find(seq_to_code));
      level = [level;dummy];
      this_run = get_run(seq_to_code,length(dummy));
      run = [run;this_run];
   end
end

level = abs(level);
S = zeros(length(level),1) ;

for mdx = 1:length(level)
   if level(mdx)
      S(mdx) = length(dec2bin(level(mdx)));
   else
      S(mdx) = 0;
   end
end

parsed = [run,S];
```

```
function [ll_q] = quantizer_ll(ll,Q,m_vec_ndx)

% Quantizes matrix ll with quantization matrix Q. M_VEC_NDX identifies
% where this quantization need be performed.
%
% see UNQUANTIZE_LL

ll_q = zeros(792,8);
offset = -8;
ll = ll* 16;

for ndx = 1:99
   offset = offset + 8;

   if m_vec_ndx(ndx)
      ll_q(offset+1:offset+8,:) = round(ll(offset+1:offset+8,:) ./ Q);
   end

end
```

```
function [out,rast] = raster(mat,in)

% Horizontal raster scans the input matris MAT of dimensions INxIN.

entry = in^2;
offset = -entry;
mat = mat.';
rast = reshape(mat,entry*99,1);

for ndx = 1:99
   offset = offset+entry;
   dummy = max(find(rast(offset+1:offset+entry)));

   if isempty(dummy)
      out(ndx) = inf;
   else
      out(ndx) = dummy;
   end

end

out = out.';
```

```
function [out] = remake_3(f_3,f_fht_ll,f_fht_lh,f_fht_hl,f_fht_hh,m_vec_ndx,in)

% Performs the inverse FHT.  F_3 is the present content of the reconstructed image.
% The next for parameters are the subbands that update this content as dictated
% by the content of M_VEC_NDX.  INxIN is the matrix dimensions.

half = in/2;

if (f_fht_ll == 0)
   f_fht_ll = zeros(99*half,half);
end

if (f_fht_lh == 0)
   f_fht_lh = zeros(99*half,half);
end

if (f_fht_hl == 0)
   f_fht_hl = zeros(99*half,half);
end

if (f_fht_hh == 0)
   f_fht_hh = zeros(99*half,half);
end

B = [1 1 1 1;1 1 -1 -1;1 -1 1 -1;1 -1 -1 1];
x = zeros(4,1);
a = zeros(4,1);
offset = -half;
out = zeros(99*in,in);
f_3 = f_3 /4;

for ndx = 1:99
   offset = offset + half;
   ost2 = offset*2;

   if m_vec_ndx(ndx)

      for rndx = 1:half

         for cndx = 1:half
            x = [f_fht_ll(rndx+offset,cndx);
```

```
            f_fht_lh(rndx+offset,cndx);
            f_fht_hl(rndx+offset,cndx);
            f_fht_hh(rndx+offset,cndx)];
         a = B \ x;
         f_3(rndx*2-1+ost2:rndx*2+ost2,cndx*2-1:cndx*2) = ...
            [a(1) a(2); a(3) a(4)];

      end

   end

end

out = f_3*4;
```

```
function [f_out] = shape(f_in)

% Shapes the 144x176 f_in into a 1584x16 matrix taken 16 rows at a time
% 1eft to right, top to bottom. (raster scans the macroblocks.)

f_out = [];

for rndx = [1 17 33 49 65 81 97 113 129]
   for cndx = [1 17 33 49 65 81 97 113 129 145 161]
      f_out = [f_out;[f_in(rndx:rndx+15,cndx:cndx+15)]];
   end
end
```

```
function [f_out] = shape_back(f_in)

% Shapes the 1584x16 matrix back into a 144x176 image.

f_out = [];
f_row = [];
offset = -176;

for i = 1:9
   offset = offset + 176;
   for rndx = ([1 17 33 49 65 81 97 113 129 145 161] + offset)
      f_row = [f_row,[f_in(rndx:rndx+15,:)]];
   end
   f_out = [f_out;f_row];
   f_row = [];
end
```

```
function [l1] = unquantize_11(11_q,Q,m_vec_ndx)

% Unquantizes 11_q with quantization matrix Q. M_VEC_NDX idenntifies where this
% unquantization need be performed.
%
% see QUANTIZE_LL

11 = zeros(792,8);
offset = -8;
```

```
for ndx = 1:99
   offset = offset + 8;

   if m_vec_ndx(ndx)
      ll(offset+1:offset+8,:) = ll_q(offset+1:offset+8,:) .* Q;
   end

end

ll = ll / 16;
```

```
function [out,vert] = vertical(mat,in)

% raster scans vertically top to bottom 99 matrices of size INxIN contained in MAT.

offset = -in;
offset2 = -in^2;
vert = zeros(offset2*99,1);

if (in == 8)
   for ndx = 1:99
      offset = offset + 8;
      offset2 = offset2 + 64;
      vert(offset2+1:offset2+64,1)=[mat(offset+1:offset+8,1);mat(offset+1:offset+8,2);
         mat(offset+1:offset+8,3);mat(offset+1:offset+8,4);mat(offset+1:offset+8,5);
         mat(offset+1:offset+8,6);mat(offset+1:offset+8,7);mat(offset+1:offset+8,8)];
      dummy = max(find(vert(offset2+1:offset2+64)));

      if isempty(dummy)
         out(ndx) = inf;
      else
         out(ndx) = dummy;
      end

   end

else

   for ndx = 1:99
      offset = offset + 4;
      offset2 = offset2 + 16;
      vert(offset2+1:offset2+16,1)=[mat(offset+1:offset+4,1);mat(offset+1:offset+4,2);
         mat(offset+1:offset+4,3);mat(offset+1:offset+4,4)];
      dummy = max(find(vert(offset2+1:offset2+16)));

      if isempty(dummy)
         out(ndx) = inf;
      else
         out(ndx) = dummy;
      end

   end

end

out = out.';
```

```
function [place,out] = zzb(bigmat,M)

% zigzag scans the 99 matrices of size MxM contained in BIGMAT.
```

```
vec = zeros(M*M,1);
loop = 0;
out = [];

for ndx = 1:M:M*99
   loop = loop + 1;
   mat = bigmat(ndx:ndx+M-1,:);
   vec(1) = mat(1,1);

   % start scan at
   x = 2;
   y = 1;
   index = 2;

   % initial scan direction
   xstep = -1;
   ystep = 1;

   % process each interior diagonal
   for i=2:(2*M-2)

      % determine diagonal length
      if (i > M)
         len = 2*M - i;
      else
         len = i;
      end

      % run the diagonal
      for j = 1:len

         vec(index) = mat(y,x);

         % move to next point
         x = x + xstep;
         y = y + ystep;
         index = index + 1;
      end

      % set up next pass
      xstep = -xstep;
      ystep = -ystep;

      if (x == 0)

         if (y <= M)
            x = 1;
         else
            x = x + 2;
            y = M;
         end

      elseif (x > M)
         x = M;
         y = y + 2;
      end

      if (y == 0)

         if (x <= M)
            y = 1;
         else
            x = M;
            y = y + 2;
         end

      elseif (y > M)
         y = M;
         x = x + 2;
      end
```

```
    end

    vec(M*M) = mat(M,M);
    dummy = max(find(vec));

    if isempty(dummy)
       place(loop) = inf;
    else
       place(loop) = dummy;
    end

   out = [out;vec];
end

place = place';
```

LIST OF REFERENCES

[1] ADM A. Clemins, "IT-21: Information Technology for the 21st Century," presented at *AFCEA West '98*, San Diego, CA, Jan. 14-16, 1998.

[2] Chief of Naval Operations, N60 - Fleet and Allied C4 Requirements Division IT-21 Homepage, http://copernicus.hq.navy.mil/divisions/n6/n60/it21/.

[3] R. Weekly, "Guaranteeing Fair Access for Multimedia Traffic Using the SMART Algorithm for Multicast ATM Connections," Master's Thesis, Naval Postgraduate School, September 1999.

[4] K. Rao and J. Hwang, *Techniques & Standards for Image, Video & Audio Coding*, Upper Saddle River, NJ: Prentice-Hall, 1996.

[5] M. Riley and I. Richardson, *Digital Video Communications*, Norword, MA: Artech House, Inc., 1997.

[6] M. H. Willebeek-LeMair, Z. Shae, and Y. Chang, "Robust H.263 Video Coding for Transmission Over the Internet," IBM Res. Rep. RC20532, Aug. 1996.

[7] S. McCanne, M. Vetterli, and V. Jacobson, "Receiver-driven layered multicast," *Proc. SIGCOMM '96*, ACM, Stanford, CA, Aug. 1996.

[8] V. Rhee and J. D. Gibson, "Rate-Constrained Two-Layer Coding of H.261 Video," *28th Asilomar Conference on Signals, Systems, & Computers*, Pacific Grove, CA, 31 Oct - 2 Nov, 1994.

[9] R. Parker, "Robust Transmission of Layered Video for Low Bit Rate Tactical Video Conferencing Applications," Ph.D. Dissertation, Naval Postgraduate School, September 1999.

[10] R. Parker, S. Skretkowicz, and M. Tummala, "Low-Complexity, Adaptive Layered Video Coder for Video Teleconferencing," To be presented at *33rd Asilomar Conference on Signals, Systems, & Computers*, Pacific Grove, CA, 24-27 Oct, 1999.

[11] C. W. Therrien, *Discrete Random Signals and Statistical Signal Processing*, Upper Saddle River, NJ: Prentice-Hall, 1992.

[12] Z. Xiong, K. Ramchandran, M. T. Orchard, and Y. Zhang, "A comparative study of DCT and wavelet based coding," *Proc. of the IEEE International Symposium on Circuits and Systems*, Monterey, CA, June 1-3, 1998.

[13] G. Kaiser, "The Fast Haar Transform," *IEEE Potentials*, vol. 17, no. 2, pp.34-36, 1998.

[14] S. McCanne, M. Vetterli, and V. Jacobson, "Low-Complexity Video Coding for Receiver-Driven Layered Multicast," *IEEE J. of Select. Areas in Comm.*, vol. 15, no. 6, pp. 983-1001, 1997.

[15] J. Gibson, T. Berger, T. Lookabaugh, D. Lindbergh, and R. Baker, *Digital Compression for Multimedia: Principles and Standards*, San Francisco, CA: Morgan Kaufmann Publishers, Inc., 1998.

[16] T. Eude, et al., "On the Distribution of DCT Coefficients," *ICASSP '94*, vol. 5, pp. 365-368, Adelaide, Australia, Apr. 1994.

[17] M. Vetterli and J. Kovacevic, *Wavelets and Subband Coding*, Upper Saddle River, NJ: Prentice-Hall, 1995.

[18] R. Frederick, "Experiences with real-time software video compression," *Proc. 6th Int. Workshop Packet Video*, Portland, OR, Sept. 1994.

[19] Telenor Research, *Video Codec Test Model, Tmn5*, Jan. 1995.

[20] V. Rhee and J. D. Gibson, "Block-Level Refinement of Motion Description in Layered H.261 Video," 29th Annual Asilomar Conference on Signals, Systems and Computers, Pacific Grove, CA, 29 Oct - 1 Nov 1995.

[21] P. Bahl and W. Hsu, "Adaptive Region-Based Multi-Scaled Motion-Compensated Video Coding for Error Prone, Bandwidth-Limited Communication Channels," *Proc. Conf. Broadband Networking Technologies, SPIE's Int'l Symp. Voice, Video and Data Communications*, Dallas, TX, Nov. 1997.

[22] Brown, T. B., Cantrell, P. E., and Gibson, J. D., "Multicast Layered Video Teleconferencing: Overcoming Bandwidth Heterogeneity," *1996 Int. Conf. on Image Processing*, Lausanne, Switzerland.

[23] H. Gharavi and M. H. Partovi, "Video coding and distribution over ATM for multipoint teleconferencing," *Proc. GLOBECOM '93*, Houston, TX, Dec. 1993.

[24] G. J. Sullivan and T. Wiegand, "Rate-Distortion Optimization for Video Compression," *IEEE Signal Processing Magazine*, vol. 15, no. 6, pp. 74-90, Nov. 1998.

[25] C. Diab, R. Prost, and R. Goutte, "Block-Adaptive Subband Coding of Images," *ICASSP*, 1990, pp. 2093-2096.

[26] ITU-T Recommendation H.263, *Video Coding for Low Bitrate Communication*, 1996.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Rd., Ste 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, CA 93943-5101

3. Chairman, Code EC 1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, CA 93943-5121

4. Prof. Murali Tummala, Code EC/Tu 4
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, CA 93943-5121

5. LCDR Robert E. Parker, U.S. Navy 1
   2452 Ponte Vedra Way
   Chula Vista, CA 91915

6. Dr. Don Gingras 1
   SPAWAR Systems Center San Diego, Code D8805
   Communication and Information Systems Department
   San Diego, CA 92152-5001

7. LT Steven J. Skretkowicz, U.S. Navy 3
   519 West "L" Street
   North Little Rock, AR 72116

66 290NPG TH 2762
6/02 22527-200 NLB

DUDLEY KNOX LIBRARY
3 2768 00357210 8